n-Cycle: a set of algorithms for task distribution on a commodity grid

Ladislau Bölöni*, Damla Turgut* and Dan C. Marinescu†
*Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816,
Email: lboloni,turgut@cpe.ucf.edu
†School of Computer Science
University of Central Florida
Orlando, FL 32816,
Email: 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 management of commodity tasks from the management of the tasks requiring specialized resources. Assuming a relative homogeneity of the commodity resource providers, the determining factor of grid performance becomes the latency 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 commodity service providers by request forwarding on the n-Cycle overlay network. We provide algorithms for task allocation and for the maintenance of the overlay network. By ensuring that the algorithms use only narrow local information, the approach is easily scalable to millions of nodes.

For task allocation algorithms in a commercial setting, fairness is of paramount importance. We investigate the properties of the proposed algorithms from the fairness point of view and show how adding several hops of random pre-walk to the algorithm can improve its fairness.

Extensive simulations prove that the approach provides efficient task allocation on networks loaded up to 95% of their capacity.

I. Introduction

The main contribution of this paper is to present and analyze the n-Cycle task distribution algorithm for commodity grids. In this introduction we motivate the need for separate treatment of commodity tasks on the grid and show the differences between the architecture of the commodity grid vs. the architecture of the grid for specialized tasks.

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. The success of the Network of Workstations (NOW, [2]) approach proved that some applications can be rewritten in such a way that they can run on commodity resources. For other applications however, such a rewrite might not be possible, or come with a significant performance penalty.

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 ([6] $\rightarrow$ [8] $\rightarrow$ [7] $\rightarrow$ [3]) 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 (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 "public computing" initiatives are exploiting the abundance of commodity resources for solving highly parallel applications. Examples are SETI@Home [19], Folding@Home the cryptographic challenges sponsored by RSA laboratories [18] or the Mersenne prime search. The Berkeley Open Infrastructure for Network Computing (BOINC, [1], [17]) 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.

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.

The tasks in the public computing approaches are *commodity tasks*. They have moderate processor and memory requirements, and they can run on any of the current generation of personal computers. In addition, public computing introduces certain simplifying assumptions, which typically do not hold in other settings: there is a single client for all tasks, there are no hard deadlines, and as there are no financial transactions involved, the accounting, fairness and security issues are of secondary importance. For instance, in the SETI@Home system, computers are "rewarded" for executing a task, but they are not penalized for accepting a task for execution and then not executing it.

We note that many high performance computing workflows contain both specialized and commodity tasks[1]. 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 [14]. The commodity components in these workflows are about the same granularity as the subtasks of the SETI@Home or Mersenne prime search, but they do not share the other simplifying assumptions of those approaches.

The rest of this paper assumes an architecture where the execution of the specialized and commodity tasks is treated separately (Figure 1). Specialized tasks are executed by queuing at the service providers, while commodity tasks are queued at the customer side and distributed to commodity service providers. The workflow engine is responsible to maintain the dependency relationships between the components of the workflow, and we allow specialized tasks to be dependent on commodity tasks.

The commodity service providers are considered essentially equivalent in performance. 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*.

The remainder of this paper is organized as follows. The architecture of the proposed system is presented in Section II. We introduce the n-Cycle overlay network, algorithms for building and maintaining it, and two distributed algorithms for task allocation in Section III. Simulation results are presented in Section IV. We overview related work in Section V and conclude in Section VI.

II. System Architecture

A. Participants and algorithm sequence

This section presents the architecture of the grid framework for the execution of commodity tasks. We start by introducing the participants:

Application Client (AC). A host which desires to run a grid workflow, where a subset of nodes are commodity tasks. The AC needs to be in contact with at least one commodity resource provider, called the *insertion point*.

[1]For instance, the authors experience with computational virology workflows shows that approximately 75% of the workflow contains image processing tasks on moderate datasets which can be written as commodity tasks [10].

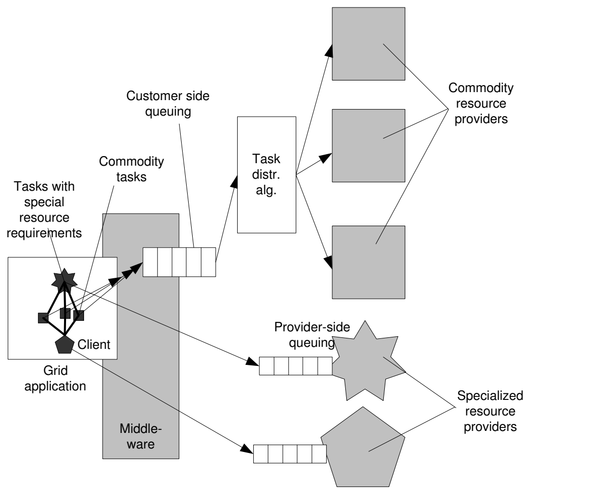

Fig. 1. The separate execution of commodity and specialized tasks of a grid workflow. Note that the queuing of specialized tasks happen at the resource provider, while commodity tasks are queued at the consumer.

Commodity Resource Providers (CRP). Computers which can execute a commodity task. The CRP's also serve as *distribution nodes*, and are able to forward task requests according to the distribution policy.

Commodity Algorithm Server (CAS). A service which provides the standard implementation of the commodity algorithms. This can be a simple FTP or HTTP based service with a specific naming convention. Alternatively, the AC can serve as the CAS.

Security and Accounting Service (SAS). A trusted web service which is used to (a) record the commitment of the CRP to execute the commodity task and (b) record the successful execution or the failure of the task. For the specialized grid, where the resource provider is well known, we frequently assume that the resource provider is trustworthy. This does not apply to the commodity grid. Public computing projects, on the other hand, frequently assume that the client is trustworthy and integrate the AC and SAS. For the commodity grid, however, we need to assume that the SAS is a trusted third party, which is not affiliated neither with the CRP nor with the AC. It is outside the purpose of this paper to discuss the actual implementation details of this service.

The general process of the algorithm is as follows:

(1) The AC formulates a commodity problem as a *task description message* TDM and sends it to one of the insertion points. The TDM contains the address of the AC.

(2) When a TDM reaches a CRP, it is either accepted for execution, or distributed/forwarded according to a *distribution policy*. If the task is accepted for execution a Task Accepted Message TAM is sent directly to the AC. We assume that every AC can accept only a single task for execution at a time.

(3) The AC, the CRP, the CAS and the SAS communicate to prepare the task for execution. If the CRP does not have a copy of the algorithm, it is downloaded from the CAS. The input data of the task is transfered to the CRP, through protocols

such as GridFTP. If the input data is very small, it can be sent in a message directly from the AC. The SAS records the successful negotiation for the start of the execution.

(4) The CRP executes the task.

(5) The CRP notifies the AC of the successful termination or failure of the task. The output data is uploaded to the AC. The SAS records the termination of the execution.

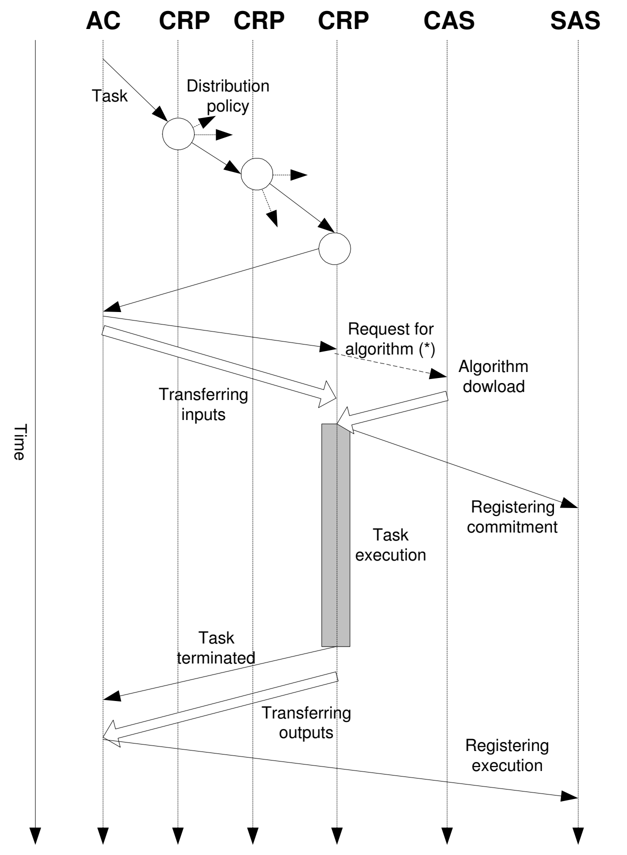

Fig. 2. The flow of the task allocation process

III. 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 overlay network and the forwarding algorithm.

A. Creation and maintenance of the overlay network

The n-Cycle algorithm creates a overlay network 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 overlay network 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. For any n-Cycle overlay network, 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.

To maintain the overlay network, we need algorithms to create a network from a set of nodes, add and remove nodes from an existing network. Algorithm 1, creates the overlay network from a set of (known) nodes. Algorithm 2 is adding a new node to the existing overlay network by making n “cuts” at random locations in the existing cycles and “splicing” the node in the cycles at these locations. The randomness of the cuts are an important part of the algorithm. Finally, Algorithm 3 removes a node from the network, by “tying together” its uplink and downlink nodes.

Algorithm 1 Creating the n-Cycle overlay network (centralized)

```
W = set of current nodes
Repeat n times
    C = copy of W
    firstNode = extract random node from C
    previousNode = firstNode
    While C not empty
        current = extract random node from C
        make current uplink from previous
        make previous downlink from current
    End While
    make firstNode uplink from current
    make current downlink from firstNode
End Repeat
```

Algorithm 2 Adding a node A to an n-Cycle overlay network

```
W = set of current nodes
For i=1 to n
    C = pick a random node from N
    D = i-th uplink from C
    make A the i-th downlink from D
    make A the i-th uplink from C
End For
W = W ⋃ A
```

Algorithm 3 Removing a node A from an n-Cycle overlay network

```
For i=1 to n
    C = the i-th downlink from A
    D = the i-th uplink from A
    make C the i-th downlink from D
    make D the i-th uplink from C
End For
```

B. Maintaining the overlay network with local information

The n-Cycle can be applied to very large networks, on the order of magnitude of several million nodes. Thus, the scalability of all the algorithm components are of importance. Primarily, we prefer fully distributed algorithms which rely only on limited local information.

Algorithms 1 and 2 require global information in the form of the set of grid nodes W[2]. We are interested in developing a completely distributed approach to creation and maintenance of the n-Cycle overlay network. First, we will not use Algorithm 1 and we will rely on adding the nodes individually to the network. Algorithm 2 for adding a node also requires the knowledge of the set of grid nodes W in order to extract the n random nodes where the insertion cuts will be made.

It is important that these cuts are randomly chosen from the *complete set of nodes* W. Let us assume that the cuts are chosen only from a set $W' \subset W$. Then, if we add a series of nodes $E = \{e_1, e_2 \dots e_n\}$ the resulting network assumes an hourglass shape, with W' being the bottleneck. The task forwarding algorithm will still work, but it will be unbalanced.

In the following, we present two fully distributed algorithms for the insertion of a new node into the n-Cycle network. The first algorithm guarantees that the node is connected through cuts which are selected from the full set of nodes W, but its complexity is linear in the size of the network. The second algorithm offers only a statistical certainty, but its complexity is logarithmic. Both algorithms exploit the properties of the n-Cycle network. For both algorithms we assume that we have an estimate of the size of the network $|W|$, this doesn't break the distributed nature of our algorithm, as overestimates are acceptable.

The first algorithm relies on a random walk on a single Hamilton cycle of the n-Cycle network. As the cycle contains all the nodes, we can pick n random cuts by randomly selecting n numbers between 0 and $|W|$, and reaching the nodes by sending a message from node to node, in the direction of the uplinks.

We can now propose the following lemma:

Lemma 1: The average number of messages exchanged exchanged during Algorithm 4 is $\frac{(2n-1)|W|}{2n}$.

We leave the proof of this lemma as an exercise to the reader. The main conclusion of this lemma is that the temporal complexity of this algorithm is $O(|W|)$, which makes the complexity of building the complete n-Cycle with repeated additions $O(\frac{|W|(|W|-1)}{2})$. Given the fact that the constant factor is proportional to the sending of a single message on the network, this complexity factor is unacceptable. For a grid of 1 million nodes and the (optimistic) assumption of 1ms processing time per message, building the network will take $0.5 \cdot 10^9$ seconds or 15.85 years.

[2]We need to note, that this is not such a great problem as if the actual forwarding algorithm would require global information. The network creation algorithm would be run only once, while the node addition will be run whenever a new node runs the system - as opposed to the forwarding rules, which need to be run several times for every task entered in the system. Furthermore, the global information required by these algorithms is a simple list of the nodes, without further status information.

Algorithm 4 Adding a node A into the n-Cycle overlay network using a random walk on a cycle

```
When node A to be inserted into overlay network W
    generate n random numbers $c_i \in \{0 \dots |W|\}$
    sort them in increasing order
    create a message $M = \{0, \{c_1, c_2 \dots c_n\}, A\}$
    send it to downlink 0
When node B receives a message $M = \{i, \{c_k \dots c_m\}, A\}$
    If $i == c_k$
        make B's k-th uplink the k-th uplink of A
        make A B's k-th uplink
        If $k < n$
            create new message
                $M' = \{i+1, \{c_{k+1} \dots c_m\}, A\}$
            send it to downlink 0
    Else
        create new message $M' = \{i+1, \{c_k \dots c_m\}, A\}$
        send it to downlink 0
```

Algorithm 5 Adding a node into the n-Cycle overlay network using a random walk on a cycle

```
When node A to be inserted into overlay network W
    For i = 0 to n
        generate $k = \lceil log(|W|) \rceil + 3$ random numbers
        $c_m \in \{0 \dots n-1\}$
        create a message $M = \{i, \{c_1 \dots c_k\}, A\}$
        send it to downlink $c_1$
When node B receives a message
    $M = \{i, \{c_j \dots c_k\}, A\}$
    If $j == k$
        make B's i-th uplink the i-th uplink of A
        make A B's i-th uplink
    Else
        create new message $M' = \{i, \{c_{j+1} \dots c_k\}, A\}$
        send it to downlink $c_j$
```

The second algorithm relies on the fact that the n Hamiltonian cycles of the n-Cycle network are independently randomized. Thus we see the downstream nodes of the insertion point to form an n-ary tree. We can now perform a random walk of k steps in this tree by randomly selecting one of the n downlink nodes. Simple probability analysis shows that the probability that a given node will not be reacheable in $\lceil log_n(|W|) \rceil + s$ steps is $(\frac{1}{e})^{n^s}$. For instance, the values for a 5-Cycle network are 0.3679 for $s = 0$, 0.0067 for $s = 1$, $1.3887 \cdot 10^{-11}$ for $s = 2$ and $5.1656 \cdot 10^{-55}$ for $s = 3$. Thus, we conclude that performing $\lceil log_n(|W|) \rceil + 3$ steps offers statistically sufficient guarantees that the new node will be inserted at a random position chosen from the whole set of existing nodes.

For this algorithm, the time complexity is much smaller, $O(n log_n(|W|))$, which leads to the complexity of the complete network building $O(n|W| log_n(|W|))$. With the previous assumptions, we calculate an average network building time of 6000 seconds or 1.6 hours.

C. Distributing tasks on the overlay network

One of the remarkable properties of the n-Cycle overlay network is that a significant majority of the nodes can be reached by only $\lceil log_n(|W|) \rceil$ 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, 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 $(1-p)^h$. 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).

Algorithm 6

```
Initially
    $w_{self} = 1$
    $w_i = 1, \ \forall i \in \{1 \ldots n\}$
When task t received by node N
    If $w_{self} == 1$
        take t into execution
        $w_{self} = 0$
    Else
        forward t to upstream node i with probability $\frac{w_i}{\sum_{k=1}^{n} w_k}$
    calculate new weight $w' = w_{self} + \frac{\sum_{k=1}^{n} w_k}{2n}$
    If $|w - w'| > \frac{1}{2n}$
        $w = w'$
        send the new weight $w$ to all upstream nodes
When execution of task t is terminated at node N
    $w_{self} = 1$
    calculate new weight $w = w_{self} + \frac{\sum_{k=1}^{n} w_k}{2n}$
    send the new weight $w$ to all upstream nodes
When weight $w$ received from i-th downstream node
    $w_i = w$
    calculate new weight $w' = w_{self} + \frac{\sum_{k=1}^{n} w_k}{2n}$
    If $|w - w'| > \frac{1}{2n}$
        $w = w'$
        send the new weight $w$ to all upstream nodes
```

D. Considerations of fairness

The architecture presented in this paper is based on the voluntary cooperation of resource providers and customers. This cooperation can be assured only if the algorithm is viewed as "fair" by the participants[3]. In this section, we consider the issues of fairness for this algorithm, and propose modifications which increase its fairness at the cost of slight reductions in efficiency.

There are two, largely independent viewpoints towards the fairness of the task allocation algorithm. From the point of view of the customers, fairness means that the servicing of a task does not depend of its customer. From a point of view of providers, fairness means that every available provider has an equal chance to service a given task. In order to assure the cooperation of both customers and providers, the algorithm needs to be satisfactory according to both measures of fairness.

1) Fairness towards customers: Fairness towards customers means that every task is serviced with the same performance parameters, independently of the customer who submitted it. The most important performance parameter, of course is whether a task is executed or rejected. A less important factor is the number of hops the task needs to travel until it is taken into execution. As long as the number of hops is small (on the order of 10-20), this factor is of little importance. For heavily loaded networks however, the additional travel time might make the difference between a task being accepted or rejected.

As there is no differentiating factor between the task description messages, the main factor is the insertion point. For networks with a single insertion point, the fairness is guaranteed by the equal treatment of the TDMs. The same consideration applies to the case where the tasks are inserted at a random point in the network, provided that the insertion point is uniformly distributed and independent of the previous tasks.

If there is a small number of fixed insertion points, tasks inserted at some points might have a greater chance to be allocated or they have to travel different distances until they are allocated. This depends on the task arrival rate at the given insertion point as well as the relative location of the insertion points in the network. Intuitively, every insertion point "fills up" the grid nodes in its vicinity, and if the insertion points are "close" to each other, the problem might be compounded.

2) Fairness towards the providers: We defined fairness towards the providers as every available provider to have the same chance of being allocated a task. In practice, it is easier to measure the number of tasks allocated to a provider during period of time. The interest of the service providers is to have as many tasks allocated as possible (given that every allocated task gives revenue).

The algorithms presented previously in this section are greedy regarding the allocation of tasks - if a TDM reaches an available host it will be immediately allocated. This leads to

[3]We note that it is out of the scope of this paper to propose methods of detecting participants who feign cooperation, or cheat through other methods.

an unbalanced (and, according to our definition, unfair) load distribution. The nodes in the vicinity of the insertion point will be almost always fully loaded, while nodes farther from the insertion point will be idle. Paradoxically, this effect is more pronounced for lighter loads. If the number of tasks is sufficiently small, the whole load can be handled by the neighbors of the insertion point, while the rest of the network would not receive any task[4].

In conclusion, the proposed algorithms are fair towards the providers only in the case in which the insertion point is a random variable uniformly distributed on the network.

3) Ensuring fairness with pre-walking: As we have seen, the presented n-Cycle algorithms are fair towards both the consumer and the provider only in the case when the insertion point for every given task is randomly and uniformly chosen from all points in the network. It is *not* enough for a consumer to choose an insertion point randomly, and later send all its tasks through that point; the random selection needs to be repeated for every task. This would require every consumer to have global information about the network.

Now, we present a simple modification in the structure of the algorithms, which will ensure fairness at the price of a small decrease in efficiency. This method exploits the built-in randomness of the n-Cycle network. We require every task description packet to perform a random pre-walk of length m before the task can be taken into execution. The pre-walk number m will be part of the task description packet. As long as $m > 0$. the packet will be forwarded randomly according to the random wandering algorithm, but it will not be allocated to any node even if the current node is free. At every forwarding the pre-walk number will be decreased by 1. The node at which the packets arrive with $m = 0$ will be called *effective insertion point*. From then on, the weighted stochastic algorithm will be followed.

With the same reasoning as in Subsection III-B, we can prove that a value of $m = \lceil log_n(|W|) \rceil + 3$ leads with a statistical certainty to a random effective insertion point.

In Subsection IV we extensively study the fairness properties of the presented algorithms with and without pre-walk.

IV. Simulation

We have used the YAES [21] simulation framework to simulate the behavior of the algorithm. Table I illustrates the input and output parameters of the simulation as specified in the YAES configuration files.

Figure 3 presents the average and maximum number of hops (top) and the total network load (below) 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 (under 10 hops), 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.

[4]The number of providers which will be allocated tasks can be determined by considering the arrival rate and distribution of the tasks, the servicing rate of N nodes and a simple queuing theoretic model.

TABLE I
Simulation parameters

Input parameters	
Number of grid nodes	10,000
Overlay network	5-Cycle
Task arrival	Poisson-distributed arrival, mean 50…200 tasks/sec
Task servicing	Normally distributed, mean 60 sec/task
Simulation time	5000 seconds
Pre-walk hops	0 and 9
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

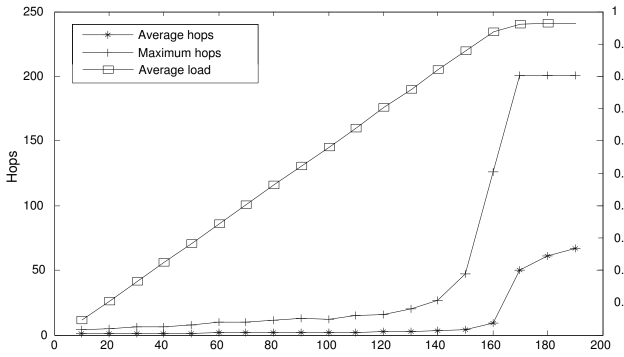

Fig. 3. Number of hops (average and maximum) and network load vs. the incoming number of tasks per second, using the weighted stochastic forwarding algorithm.

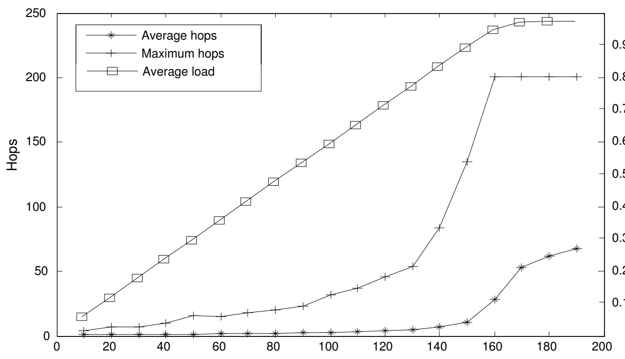

Fig. 4. Number of hops (average and maximum) and network load vs. the incoming number of tasks per second, using random forwarding on the n-Cycle overlay network.

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 $log_N(|W|)$ 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 4 presents the random walking algorithm. For light loads, this algorithm also shows very good results (due to the randomizing nature of the N-Cycle network). However, for greater loads, the maximum number of hops start to increase. For instance, at a load of 90% the maximum will be as high as 75 hops vs. about 10 hops for the stochastically weighted algorithm.

Fairness measures

In a separate series of measurements, we have measured the fairness towards the providers in function of the number of tasks arriving to the network. The measurements were performed by counting the number of tasks executed by every host. These values were then sorted and four values picked at the minimum, maximum, 5% and 95% levels. The reason for plotting the intermediate values is to filter out providers having special position in the network. For example, for a single insertion point network, the insertion point has a special situation, given that all incoming tasks are passing through it.

Figure 5 shows the results of the measurements for the case of a single insertion point. As expected, the maximum value shows that the insertion point will achieve 100% load. The bottom 5% has no tasks allocated for task arrival rate as high as 40 tasks/seconds. The gap between the four measurements is higher at low loads, and lower at high loads when even providers far from the entry point will be allocated tasks. This measurement validates our prediction that for single insertion point method leads to unbalanced and unfair distribution of tasks.

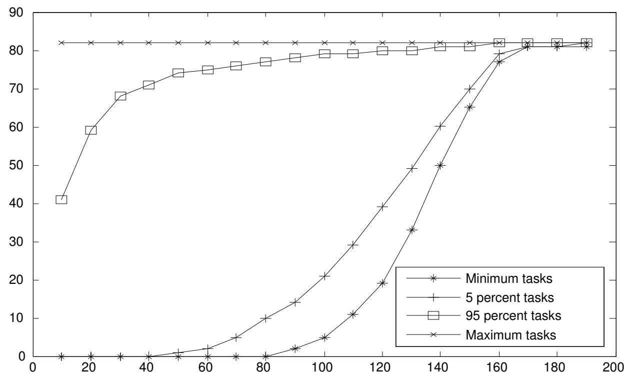

Fig. 5. Fairness in terms of tasks allocated to providers. Single insertion point, no prewalk.

Figure 6 presents the measurements for the case of a random insertion point. Again, the simulation results match the prediction, the number of tasks executed by the nodes being in a relatively narrow range, without standout values. As an observation, the reason of the spread in the values is due to the inherent randomness (Poisson arrival, random insertion point, normally distributed execution time) and the limited timeframe of the simulation. Simulated over longer timeframes, these values are converging to a single line. This is basically the ideal fairness, but as we stated before, it requires global information about the network to prepare a proper random insertion point.

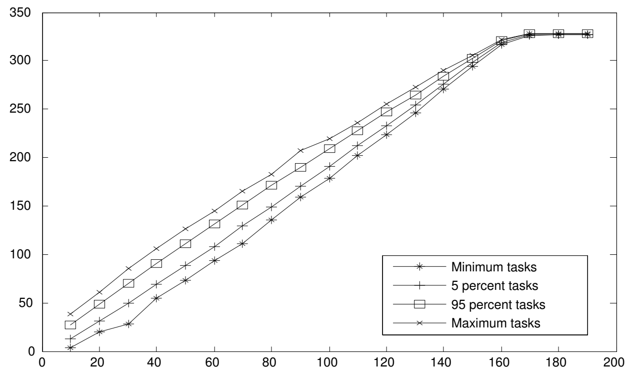

Fig. 6. Fairness in terms of tasks allocated to providers. Random insertion point, no pre-walk.

Figure 7 shows the measurements for a single insertion point and the task distribution algorithm including a 9 hop pre-walk. The value of 9 is the empirically obtained value of $\lceil log_n(|W|) \rceil + 3$ for $|W| = 10000$ and $n = 5$. We should note the resemblance of the diagram to Figure 6. We conclude that a pre-walk with sufficient number of hops achieves the same results as the random insertion point approach, while still requiring only local information.

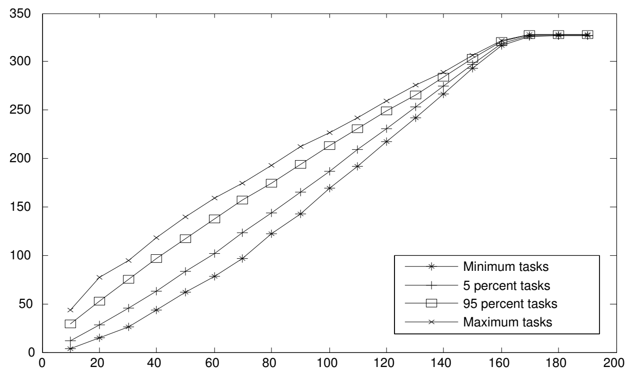

Fig. 7. Fairness in terms of tasks allocated to providers. Single insertion point and 9-hop pre-walking.

V. Related work

This paper proposes an architecture where the commodity tasks are allocated on a grid by the forwarding of the requests. Similar designs are proposed in [9], [4]. The Wire Speed Grid Project [20], proposes an architecture in which the task allocation is performed hardware accelerated on the network routers.

The algorithms presented in this paper have their closest relatives in the class of distributed algorithms which create and exploit an additional graph structure, built on top of the existing, fully connected internet (often called *overlay network*).

One of the most important classes of overlay networks are Distributed Hash Tables (DHT). These networks store pieces of data with their associated unique key. Every key and the associated data is mapped to a certain host, which is normally not known to the user. Data can be inserted and retrieved from a DHT without knowledge on where it will actually be stored - in fact, it is possible that the location of data will change as hosts join and leave the DHT. A number of DHT architectures were proposed such as CAN [11], Chord[13], Pastry [12], and Tapestry [15]. For most of these networks every node maintains $O(log(N))$ neighbors and a message can be routed in $O(log(N))$ hops.

The properties of a DHT allow us to use it as the basis for a resource discovery and allocation framework. An example of this is the Self-Organizing Flock of Condors project [5] which is augmenting the Condor program with a DHT based on the Pastry overlay.

The CCOF (Cluster Computing on The Fly) project [16] implements a system in which idle cycles are harvested from a collection of computers. The system employs community based overlay networks, which allow hosts to dynamically join and leave. For the actual resource allocation step, a variety of search algorithms were implemented and measured, the most complex being Advertisement Based Search and Rendezvous Point Search.

Let us now succintly state the position of the n-Cycle model in respect to the other approaches. The n-Cycle model builds an overlay network, which, however, is not a DHT. The request distribution algorithm is strictly unicast, and the algorithm requires exclusively local information both for the actual forwarding and the maintenance of the overlay network. The grid model we assume is one of commodity tasks and commodity resource providers. We argued that under these assumptions, it is more efficient to queue the tasks on the consumer side then on the provider side. This differentiates us from models which are based on provider side queuing such as the one provided by Condor, which are more appropriate for specialized resources. The system we are considering assumes some sort of payment for the services provided, and that even if the same host might alternatively provider and customer, there is no guarantee that the credits will be equalized. The consequence of this is that the assumption of "long term fairness" is not appropriate here and the fairness of the task allocation needs to be guaranteed with explicit techniques.

VI. Conclusions and future work

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

Our future work will include a more extensive, queueing theory based analysis of the algorithm. For a practical deployment it is of a particular interest to develop protocols for controlling the task arrival rate. This will likely raise interesting problems regarding the fairness towards the customers. Another natural extension is to allow for some level of heterogeneity in the grid both in terms of the providers as in terms of the tasks.

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] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, 1995.

[3] M. Baker. Ian Foster on recent changes in the grid community. URL `http://dsonline.computer.org/0402/d/o2004a.htm`.

[4] B.Liljeqvist and L.Bengtsson. Grid computing distribution using network processors. In *Proc. of the 14th IASTED Parallel and Distributed Computing Conference*, Nov 2002.

[5] A. R. Butt, R. Zhang, and Y. C. Hu. A self-organizing flock of Condors. In *In Proceedings of IEEE/ACM Supercomputing 2003, Phoenix, AZ*, November 2003.

[6] I. Foster and C. Kesselman, editors. *The Computational Grid: Blueprint to a New Computer Infrastructure*. Morgan-Kauffman, 1998.

[7] 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`.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[9] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. In *Proceedings of the International Workshop on Grid Computing, Denver, CO*, November 2001.

[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(7-8):738–758, 2003.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 161–172, 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 149–160, 2001.

[14] 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.

[15] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.

[16] D. Zhou and V. Lo. Cluster computing on the fly: Resource discovery in a cycle sharing peer to peer system. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 66–73, 2004.

[17] Berkeley Open Infrastructure for Network Computing. URL `http://boinc.berkeley.edu/`.

[18] RSA Challenge. URL `http://www.rsasecurity.com/rsalabs/challenges/`.

[19] SETI@Home project. URL `http://setiathome.ssl.berkeley.edu/`.

[20] The Wire Speed Grid project. URL `http://www.ce.chalmers.se/staff/labe/Wire Speed Grid Project.htm`.

[21] YAES: Yet Another Extensible Simulator. URL `http://netmoc.cpe.ucf.edu/Yaes/Yaes.html`.