

# Relative Autonomous Accounting for Peer-to-Peer Grids

Robson Santos

robson@dsc.ufcg.edu.br

Alisson Andrade

aandrade@dsc.ufcg.edu.br

Walfredo Cirne

walfredo@dsc.ufcg.edu.br

Francisco Brasileiro

fubica@dsc.ufcg.edu.br

Nazareno Andrade

nazareno@dsc.ufcg.edu.br

Universidade Federal de Campina Grande

58.109-970, Campina Grande, PB, Brasil

Phone: +55 83 3310 1365

## ABSTRACT

We here present and evaluate *relative accounting*, an autonomous accounting scheme that provides accurate results even when the parties (consumer and provider) do not trust each other. Relative accounting relies on the observed *relative performance* among the parties. As such, the basic requirement to use it is that resource consumers must also be resource providers. Relative accounting is totally autonomous in the sense that it uses only local information, i.e. there is no exchange of information between the parties. This allows for the deployment of the autonomous accounting without requiring any sort of identification infrastructure, such as certificate authorities. Not requiring trust or sophisticated infrastructure makes relative accounting a perfect fit for peer-to-peer grids, which aim to scale much further than traditional grids by allowing free unidentified entry into the grid. Our results show that relative accounting performs very close to a perfect accounting, whose implementation is infeasible in most systems, including those we target. Relative accounting was developed to work with OurGrid, a peer-to-peer grid in production since December 2004, but it can also be used in other peer-to-peer grids.

## KEYWORDS

Accounting, resource usage, reputation model, peer-to-peer, grids.

## 1. INTRODUCTION

Computational grids have appeared about a decade ago with the promise of delivering great amounts of computational power to parallel applications by enabling the seamless sharing of resources spread among different administrative domains. The grid promise has latter evolved into enabling on-demand access and composition of computational services located throughout the globe. Meanwhile, the first grid systems went into production: TeraGrid, CERN's LCG, OurGrid, Grid3, and many others. This is not to say, however, that all grid problems have been solved. We are still far from that. In

particular, we would like to draw the reader's attention to *accounting* and *scalability*; two issues in grid computing that clearly demand further research.

Accounting aims to measure resource usage. A grid provider typically wants to gauge how much resources a given client consumed, either to control such usage or to charge for it. Since a grid comprises multiple administrative domains, this raises a question of trust: should the consumer believe the accounting gauged by the provider? Furthermore, finding a metric to represent resource usage is not a simple task, as we shall see in Section 2.

Scalability is also an issue because grids today require complex installation and set-up, and, more importantly, demand human negotiation among the different administrative domains to define how resources are going to be accessed and shared. This is the case because grids aim to provide sophisticated services, typically with some guarantee on what users can expect of such services [15]. Using peer-to-peer techniques has appeared as a promising approach to foster grid scalability [15]. In fact, there are now many peer-to-peer grid projects, e.g. [3] [10] [11] [19] [22] [26], and the first peer-to-peer grids are also going into production. Naturally, peer-to-peer grids do not yet provide services as rich as traditional grids. Application execution (i.e. CPU sharing) appears to be the first service being tackled in peer-to-peer grids. On the other hand, peer-to-peer grids aim to scale for the planet! They try to create a global scale resource-sharing network, so that anyone who requires large amounts of computing power needs just to connect to this network and use the resources that are available.

But notice that accounting becomes an even tougher problem for peer-to-peer grids. In traditional grids, where participants are known and authenticated, there is typically reasonable trust among the administrative domains, and accounting solutions can rely on such trust. On a peer-to-peer grid, on the other hand, participants can freely join the system, without providing a strong link to their real meat-space identities [12]. Therefore, participants are essentially anonymous, and no solutions (including accounting) can be built assuming trust among the parties.

Yet, accounting may be even more important for peer-to-peer grids than it is for traditional ones. This is because free-riding is a common behavior in peer-to-peer systems [1] [2] [8] [17] [21] [25]. A free-rider is a peer that only consumes resources, never contributing back to the system. If there is a non-zero cost for providing resources to the grid, and free-riders get the same benefit from the system as peers that contribute to the system, then it is in the best interest of peers to free-ride and the system may collapse [7]. Coping with free-riding in a peer-to-peer system typically involves some reciprocation strategy, on which a given peer tries to help peers that have helped it in the past [2] [4] [5] [6] [8] [13] [21]. Therefore, it is crucial for a peer to be able to gauge the

contribution another peer provided to it, a requirement that, in a CPU sharing peer-to-peer grid, translates into a strong need for accurate accounting.

We here present *relative accounting*, an autonomous accounting scheme that relies on the relative power of each peer to provide accurate accounting for peer-to-peer grids. We rely on the fact that, in a peer-to-peer grid, resource consumers are also resource providers to autonomously evaluate the relative performance of the parties, and then use this information to normalize the time a resource has been allocated by a provider to a consumer. Relative accounting is totally autonomous in the sense that it can be fully evaluated using only local information. In fact, the best and simplest way to avoid the need of trust relationship among the parties in the system is autonomously accounting the resource allocations. As a result, we do not need a complex infra-structure for its deployment as cryptographed certificates and trusted auditing peers, because no accounting information is exchanged among the participants. The accounting values are independently maintained by each peer locally. In short, we do not introduce any need for trust raising entities, such as certificate authorities, thus not precluding peer-to-peer grids from reaching very large scale. Our efforts extend initial results presented in [23] and are part of the OurGrid project developed in a collaboration between the Universidade Federal de Campina Grande (UFCG) and Hewlett Packard (HP) [22]. Relative accounting was conceived for OurGrid, but it can be useful for other peer-to-peer systems on which consumers also provide resources.

The rest of this paper is structured as follows. In Section 2 we describe the accounting problem and discuss some possible approaches to solve it, including the definition of both a perfect and a simplified time-based accounting. We also discuss the difficulties and drawbacks of using these schemes, motivating the introduction of our relative autonomous accounting. Section 3 presents our autonomous accounting scheme that uses the relative computational power of peers to accurately account resource utilization. Then, in Section 4, we analyze how well our autonomous accounting performs, comparing it against the perfect and the time-based schemes. In Section 5 we discuss related works. Section 6 concludes the paper with our final remarks.

## **2. THE RESOURCE ACCOUNTING PROBLEM**

We assume that grid *jobs* are composed of *tasks* that run in parallel. The accounting of a resource allocation is a function of (i) the processing power that the computing resource provided (or used, depending on the viewpoint of who is calculating the allocation's value) is able to deliver for a particular task at a given point in time, and of (ii) the time interval during which the resource is provided (or used). Assuming that  $power_p(\tau, t)$  is the amount of processing power a computing resource  $P$  is able to deliver to a task  $\tau$  at a

particular instant of time  $t$ , the *perfect accounting* of running task  $\tau$  on processor  $P$  starting at some time  $t_i$ , for the interval of time  $\Delta t$  would be  $AP(\tau, P) = \int_{t_i}^{t_i+\Delta t} power_P(\tau, t) dt$ .

As can be seen, the difficulty in providing accurate accounting lies in determining  $power_p(\tau, t)$  for each instant of time  $t$ . Assessing  $power_p(\tau, t)$  is complicated by two main factors. First, since computer performance depends on the application, the same computing resource may deliver different performance to different tasks even when the resource is subject to the same competing load and allocated for the same amount of time. For instance, a resource with a fast CPU but with a slow disk may be more valuable to CPU-intensive tasks than to I/O-intensive ones. Second, this information has to be inferred autonomously by the consumer peer, without relying on information of untrustworthy peers. In particular, the consumer cannot trust the  $power_p(\tau, t)$  informed by the provider of  $P$ .

Consequently, it is unlikely that such a perfect accounting can be built, unless some restrictions are imposed (for instance, by using dedicated resources to run only a priori analyzed tasks). Alternatively, one can try to develop practical accounting schemes that are good approximations to the perfect scheme. A simple approximation would be to consider that all computing resources have the same power at any given time. For instance, one can assume that all resources have power 1 at all times and simply use the time a resource has been made available to a remote peer as the accounting of its resource allocation. Using such time-based scheme, the accounting of running task  $\tau$  on processor  $P$  starting at some time  $t_i$ , for the interval of time  $\Delta t$  would produce  $AT(\tau, P) = \Delta t$ .

Since the time interval a resource is made available is easily and autonomously computed by both consumer and provider, this approach can easily be deployed on grids in which peers do not trust each other. However, such a time-based accounting does not take into account the amount of work done by the resource. Thus, peers whose resources take longer to process tasks will get more benefits from the grid than those whose resources are faster and execute the same tasks faster. In Section 4 we discuss the negative impacts that such an approach may bring to the grid as a whole.

### 3. ACCURATE AND AUTONOMOUS ACCOUNTING

In order to circumvent the problems just described, we have developed a new approach to do accounting that considers the relative power of the peers in the grid. In a peer-to-peer grid, a peer is typically responsible for a site (i.e. resources under a single administrative domain, normally in a LAN). As such, reputation models would rank peers

rather than resources, and hence we make relative accounting calculate the accounting per peer.

A peer has local resources that are used to run some tasks of its jobs and from remote peers' jobs. We take advantage of this fact to define a new metric, named relative power *rpower*, which is used to account for task executions. The relative power of a remote peer is a measurement of how much faster or slower this peer is, when compared to the local peer. That is,  $rpower_B(A)$  is the relative power that a peer  $A$  is currently able to deliver, as perceived by a peer  $B$ . It is estimated by peer  $B$  as the average execution time of the tasks  $B$  ran locally, divided by the average execution time of the tasks  $B$  ran on  $A$ . That is,

$$rpower_B(A) = \frac{\text{mean}\{\text{exectime}(\tau_B(B))\}}{\text{mean}\{\text{exectime}(\tau_B(A))\}}, \text{ where } \{\text{exectime}(\tau_B(B))\} \text{ denotes the execution}$$

time of the tasks that  $B$  ran on  $B$  (i.e. locally), whereas  $\{\text{exectime}(\tau_B(A))\}$  denotes the execution time of the tasks that  $B$  ran on  $A$ . For example, if peer  $A$  runs  $B$ 's tasks in half of the time taken by  $B$  itself,  $rpower_B(A) = 2$ . Also, for any peer  $A$ ,  $rpower_A(A) = 1$ .

The relative accounting of running task  $\tau$  from peer  $B$  on peer  $A$ , starting at some time  $t_i$  for the interval of time  $\Delta t$ , is defined in  $B$  as  $AR_B(\tau, A) = \Delta t \times rpower_B(A)$  and in  $A$  as  $AR_A(\tau, A) = \Delta t \times rpower_A(A) = \Delta t$ .

Furthermore, as both the consumer and the provider can measure the time a resource has been in use as well as evaluate their power relative to that of the resource provider, our scheme is completely autonomous and does not require peers to share information. On the other hand, as we rely on the average execution time of tasks, our scheme works better when tasks demand similar amounts of processing. Heterogeneity in execution time of tasks may introduce errors in relative accounting. However, as we shall see in Section 4.5, this error is not significant in the representative settings we have analyzed.

## 4. PERFORMANCE EVALUATION

We have used simulations to analyze the different accounting schemes. We wrote an event-based simulator for the peer-to-peer computational grid described in Section 4.1 and implemented the three accounting schemes discussed: perfect, time-based and relative accountings. Note that perfect accounting can be easily implemented in the simulator. Since the simulations run in a completely virtual environment created by the simulator, we introduced an invisible and omniscient "god" entity that looks inside each peer, extracting the computational cost of each task.

### 4.1. Peer-to-Peer Grid Model

Typically, in a peer-to-peer computational grid, each peer represents an administrative domain. Each peer controls the access to a set of local computational resources and plays

two roles: (i) it may work as a provider, allowing its (otherwise) idle resources to execute other peers' tasks, and (ii) it may work as a consumer, using its local resources and any idle resources from other peers to execute its own tasks. However, as pointed out before, free-riding is a common behavior in peer-to-peer systems [1] [2] [8] [17] [21] [25]. Free-riding is particularly undesirable in peer-to-peer computational grids because the benefits that users get from this kind of system are proportional to the amount of resources available.

We thus assume that there is a mechanism to discourage free riding. More precisely, we assume that peers reciprocate resource allocations using the Network of Favors (NoF) [4] [5] [6]. NoF is an efficient and lightweight reputation-based resource allocation mechanism designed to promote contributions of resources in a peer-to-peer grid. In NoF, the more resources a peer provides, the more likely it will receive resources from other peers when it needs them. It has been used to promote resource sharing in OurGrid, a free-to-join peer-to-peer grid that is in production since December 2004 and that currently combines around 300 machines spread in about 15 sites [12] [22].

In NoF terminology, resource allocation is called a *favor*. Resource providers give favors and resource consumers receive them. For a peer  $A$ , the reputation ranking of a peer  $B$  is represented by the balance of favors they have exchanged, i.e. the amount of favors  $A$  has received from  $B$  minus the amount of favors  $A$  has provided to  $B$ . Also, to prevent id-changing attacks, balances are always kept non-negative. (A peer performs an id-changing attack by leaving the system and immediately re-entering it using a new identification, thus resetting its balance to zero.) When a peer has idle resources, it provides them to the grid. If two remote peers request these resources, then the remote peer with the highest local balance gets the resources. However, users of the local peer always have priority over those from remote peers. Thus, whenever tasks from local users are submitted, any tasks from remote users are aborted. This simple set of norms makes it in the best interest of peers to provide to the grid as much computational power as they can [4] [5] [6].

Note that NoF is completely decentralized and autonomous, in the sense that there is no exchange of reputation information among peers. In particular, for any 3 peers  $A$ ,  $B$  and  $C$ , the reputation of  $C$  in  $A$ 's eyes bears no relation with the reputation of  $C$  in  $B$ 's eyes. The reputation ranking is calculated using only local information about the favors. However, for the NoF to work well, it assumes an accurate accounting mechanism as it must reflect the actual amount of resources that are consumed and provided by peers. It is important that the accounting is also autonomous, so that the NoF can continue to be deployed without requiring any special agreements between peers or access to specialized certification services.

We also assume that it is possible to evaluate whether the computation sent from one peer to another was correctly executed. Naturally, computation should only be accounted when the results are indeed correct. This issue can be dealt with by using the credibility-based fault tolerance scheme proposed by Sarmenta [24].

Finally, note that we neither model machine failures nor network delays. Failures can be easily dealt with. If a machine fails while processing a given task, no useful work was done and the task will have to restart elsewhere. Therefore, the allocation of such machine to run the task should be accounted for as producing zero work. In particular, the gauging of mean task execution time should ignore failed local executions. Since the grid middleware must know when tasks fail (as to resubmit them), this is straightforward to implement.

Note that network delays should not be considered when accounting the execution time of a task. For instance, in OurGrid a task is executed in three steps: the initial step typically performs the upload of input data and executables; then a remote step is performed to execute one or more programs at the remote machine; a final step concludes the execution of the task, normally collecting any output generated in the previous step. Therefore, the CPU accounting should only consider the time to execute the remote step.

However, although accounting only for CPU utilization is adequate for CPU-intensive applications, this may be unfair with data-intensive ones. Accounting for data transfer is straightforward, since it only requires counting the number of bytes transferred. What complicates matters is how to convert data transfers favors into CPU favors and vice-versa, i.e. how to combine both accountings into a single value. This is a key issue for using the Network of Favors.

A straightforward solution to this problem is to arbitrate a fixed “exchange rate” to perform this conversion. Alternatively, one can use a multiple service economy, in which each site has its own exchange rate that can vary over time. Our initial study of such a system indicates that the Network of Favors can be extended to this setting with relative ease [20].

## **4.2. Simulator and Scenario Generator**

In order to analyze the performance of the three accounting schemes studied in this paper, we have developed an event-based simulator for the peer-to-peer grid model described above and a generator to produce the statistically meaningful scenarios necessary for the analysis of those schemes (see Section 4.4). Our simulator is open source and is available for downloading in the On-going Work Section of the OurGrid site, at <http://www.ourgrid.org/twiki-public/pub/OG/OurOngoingWork/PublicAccountingSimulator.zip>.

The scenario generator is a program that receives a description of a setting in an XML file and generates a random instance (scenario) that fits the specifications of that setting. A setting is a set of parameters and their range of values that configure the grid model that will be executed by the simulator. That is, given a setting description, the scenario generator produces a scenario by randomly choosing one value from the ranges of values for each parameter in the setting. The simulator then receives the scenario, instantiates its entities (peers, machines, jobs, tasks) and simulates it. When the simulation finishes, the simulator generates XML files describing the results of the simulation (how jobs, peers and favors have finished). For instance, in the job result file, there is information on the simulation time at which the job was submitted and at which it has finished, whereas in the peer result file, there is information on the number of resources donated and consumed throughout the simulation.

In our analysis, we have varied the following parameters for creating different settings:

- *Size of the grid:* this parameter sets the number of peers that comprise a grid. We have experimented grids with 10 to 100 peers. The experiments have shown that varying the number of peers does not affect the accounting results. This is expected as previous results have shown that, at least up to 10,000 peers, reciprocation-based peer-to-peer grids display very little sensitivity to number of peers in the system [6]. In this paper, we show the results for grids with 15 and 20 peers, which is about the number of peers currently present in OurGrid [12] [22].
- *Number of resources per peer:* this parameter sets the number of resources that comprise a peer and allows us to create peers of different sizes and total power.
- *Mean resource power:* varying the mean power of the machines in each peer allows us to verify the fairness of the accounting schemes when dealing with resources with different computational power.
- *Number of tasks per application:* this parameter sets the size of the applications submitted to the grid and allows us to analyze the performance of the accounting given applications of different sizes.
- *Size of tasks:* with this parameter we can verify the behavior of the accounting in face of homogeneous applications (computational cost of all tasks is the same) and heterogeneous ones (there is considerable variation among the computational cost of tasks).
- *Number of jobs submitted per peer:* this parameter allows us to control the simulation total time by increasing or decreasing the number of applications submitted to the peers.

With those parameters we could build settings that vary both in the grid composition (peers with slow and fast resources) and in the workload submitted to the participant peers (large/small homogeneous/heterogeneous applications).

### 4.3. Metrics

In order to analyze the simulations, we have defined two metrics: mean response time of jobs and favor ratio. As grid users are very much concerned about diminishing the turnaround time of their jobs, we observe in our simulations the mean response time of jobs. The response time is the time elapsed between the submission of a job and its completion. The relation between mean response time and accounting is explained by the way the NoF works: peers that provide more processing power to the grid obtain more resources when they make a request. Therefore, peers that contribute more should have smaller mean response times for the execution of their jobs.

Another important metric we have used to help us understanding the behavior of the system is the *favor ratio*,  $fr(A) = \frac{resourcesConsumed(A)}{resourcesDonated(A)}$ , where  $resourcesConsumed(A)$

is the total processing power consumed by peer  $A$  from other peers in the grid, and  $resourcesDonated(A)$  is the total processing power provided by  $A$ , calculated using the real computational cost of the tasks processed by each peer. Assuming perfect accounting and that the system has contention for resources, the NoF makes favor ratios tend to 1 for all peers in the grid, thus ensuring *fairness* [5] [6]. However, in reality, it is not possible to implement perfect accounting. Therefore, we are interested on how “implementable accounting schemes” affect the NoF fairness.

Note also that, in practice (and in our simulator), another factor that influences the favor ratio is the abortion of tasks. A peer aborts all foreign tasks when it receives a local job. When this happens, the consumer sees no value in the incomplete favor, and the provider gains no credit in its eyes. Therefore, even for perfect accounting, *favor ratios* tend to converge to a value smaller than 1 in our simulator (and in practice).

### 4.4. Statistical Sampling

As evaluation scenarios are generated randomly, each setting evaluated must have enough samples for the results to be statistically meaningful. To do that, we chose a confidence level of 95%. Given our confidence level, in the worst case we needed to collect at least 280 samples. To keep it simple, we executed 280 scenarios for all settings evaluated.

As it would take too long to execute all simulations in a desktop computer, we used the OurGrid itself to access hundreds of machines in different administrative domains and run the simulations much quicker. Overall, using OurGrid enabled us to run simulations

about 40 times faster than using a single machine.

#### 4.5. Performance Evaluation

This section presents the results of the experiments we conducted in order to analyze the performance of the three accounting schemes. To do that, we have defined five settings (each corresponding to 280 simulated scenarios). In all scenarios simulated, we ensured that the workload submitted to the grid was high enough to saturate the grid, creating contention for resources. This is important because if there is a surplus of resources in the grid, then prioritizing peers has no clear effect, as everyone gets the resources they need from the system. As we are interested in observing how the NoF behaves with different accounting schemes, the system must have, on average, more than one peer interested in the resources available. To cause such contention, the inter-arrival time of jobs at peers follow a uniform distribution whose average is calculated by dividing the total amount of work required to execute an average job by the sum of the power of all the resources in an average peer. Therefore, due to the probabilistic nature of job arrival, peers are sometimes idle and sometimes need to ask for more resources from the grid.

The first setting describes a grid consisting of peers that contain resources with different powers, to which tasks of different sizes are submitted. This setting corresponds to *a frequent case we expect to find in practice*, in which the grid is formed by peers with heterogeneous resources, and jobs vary in size.

The second setting describes a situation in which peers have the same number of resources, those resources have the same power, but tasks vary in size according to uniform distributions. Although unlikely in practice, this represents a *disadvantageous setting for relative accounting*, because the variation in the size of tasks introduces errors in the estimation of the relative power of peers. Moreover, since resources have the same power, the time-based approach should have a performance equivalent to the perfect accounting.

The third setting is the same as the second, except that the sizes of the tasks vary according to an exponential distribution, instead of a uniform one. That is supposed to make the setting *even worse for the relative accounting*, although it makes this setting more unlikely to appear in real grid systems.

The fourth setting shows an *accounting attack*, a situation in which half of the peers behave maliciously. They attack the accounting mechanism by pretending to have twice the number of resources that they actually have by running two tasks per resource simultaneously. This setting analyzes the robustness of the accounting schemes.

Finally, the last setting evaluates the performance of the three accountings in a grid in which the *total power of the participant peers are different*. This is to observe how the

total power of the peers affects the behavior of the accounting schemes and, consequently, the exchange of resources among them.

#### *4.5.1. Setting 1: Common Case*

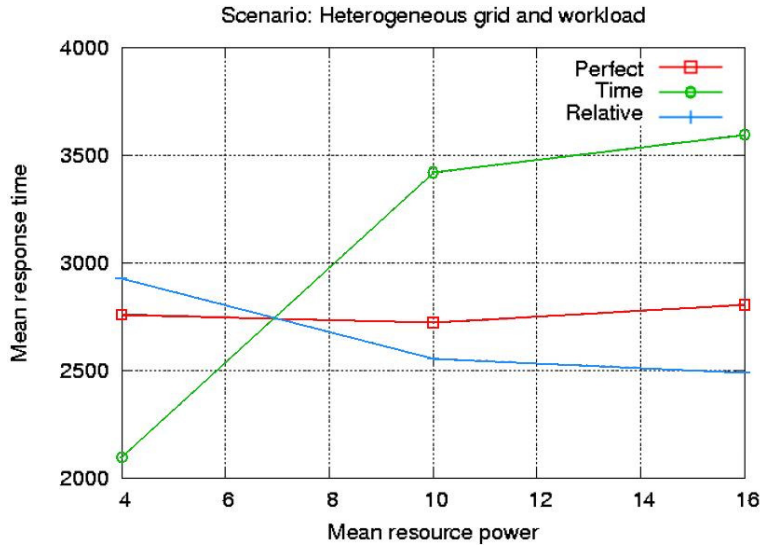
In this setting we try to capture the heterogeneity of resources and tasks found in a grid. Thus the power of the resources and the size of the tasks vary. However, we keep the total computing power owned by each peer fixed. This means that peers are able to contribute the same amount to the grid and therefore should get the same amount of resources when they make a request. With perfect accounting, this should translate into all peers having similar mean response times and a favor ratio that is close to 1.

We define three types of peers: those with mean resource power 4, 10, and 16. In the simulations we have a grid with 15 peers, 5 peers of each type. The power of a resource in these types is given by the uniform distributions  $U(2,6)$ ,  $U(5,15)$  and  $U(8,24)$ , respectively. (Kee et al. show that it is reasonable to assume that the grid processors follow a uniform distribution [18]). In addition, the sum of the power of all resources in each peer is 250. Note that this implies that peers have different number of resources.

All jobs submitted to the peers in this setting consist of 250 tasks, each task with its size varying according to a uniform distribution  $U(200,600)$ . As to saturate the peers (on average), the inter-arrival time of jobs at each peer follows the uniform distribution  $U(1, 799)$ , and 1,000 jobs were submitted to each of them.

Figure 1 shows the average response time for each type of peer (mean resource power 4, 10 and 16) in the simulations. The fact that peers are able to get the same amount of resources from the grid if everyone accounts correctly is reflected in the very small variation in the mean response times when using the perfect accounting. The reason that there is any variation at all is that peers with slower resources are more prone to abort tasks of other peers running in their resources, as there is a higher probability that local jobs will be submitted during the execution of these tasks.

On the other hand, the time-based accounting is strongly biased towards peers with slower resources. When a resource takes longer to process a task, the favor is (unfairly) assigned a higher value. Moreover, this difference is very high: peers with the slowest resources have their jobs processed on average 71.6% faster than those with the fastest resources.



**Figure 1. Power vs. mean response time for setting 1**

Finally, the relative accounting is slightly biased towards peers with faster resources, but it is much closer to the perfect scheme. Under this scheme peers with faster resources have mean response times 17.6% better than peers with slower ones.

Table 1 shows the favor ratios for each type of peer and each accounting scheme. These results confirm the previous analysis. The mean favor ratio for the perfect and the relative accountings are very close, with the relative accounting being slightly more favorable to peers with faster resources, when compared to the perfect accounting. On the other hand, for the time-based approach there is an enormous discrepancy between the mean favor ratio of the peers with mean resource power 4 and 16, with the peers with slower resources being better off than peers with faster resources.

**Table 1. Favor ratio for peers in setting 1**

	Mean (Std Dev) of $fr$ for peers with power 4	Mean (Std Dev) of $fr$ for peers with power 10	Mean (Std Dev) of $fr$ for peers with power 16
<b>Perfect</b>	0.908 (0.114)	0.981 (0.099)	0.994 (0.101)
<b>Relative</b>	0.872 (0.122)	0.995 (0.079)	1.020 (0.075)
<b>Time</b>	1.051 (0.084)	0.866 (0.095)	0.830 (0.100)

This setting gives evidence that the relative accounting is suitable for a peer-to-peer grid, as it has behavior close to that of a perfect accounting. Moreover, the small bias towards peers with faster resources has the side-effect of providing an incentive for peers to make the fastest possible resources available. Since the bias is not large, we believe it should not be enough to put off peers that cannot afford to own fast resources.

#### 4.5.2. Setting 2: Unfavorable Case (uniform distribution)

In this setting, we have 15 peers of the same type, each peer with 25 resources with computational power equal to 10. However, the jobs submitted to each peer contain tasks that are heterogeneous in size, varying according to a uniform distribution  $U(100,700)$ . Each peer received 1,000 jobs consisting of 250 of these tasks. We created contention in the grid by submitting those jobs with inter-arrival times following a uniform distribution  $U(1, 799)$ .

**Table 2. Response time and favor ratios for peers in setting 2**

	<b>Mean (Std Dev) of Response Time</b>	<b>Mean (Std Dev) of Favor Ratio</b>
<b>Perfect</b>	2,486.7 (2,610.7)	0.9645 (0.0851)
<b>Relative</b>	2,515.3 (2,664.6)	0.9556 (0.0971)
<b>Time</b>	2,486.6 (2,610.7)	0.9645 (0.0851)

This setting is a special case in which accounting a favor correctly is equivalent to accounting the time the task took to be processed in a resource. In such a case, we expect to see the best performance of the time-based scheme. Table 2 shows the mean response times and favor ratios for each type of accounting scheme. As expected, the perfect and the time-based accountings have essentially identical performance.

The relative accounting results are a bit worse than the results of the other two schemes. However, when we quantify how worse they are, it gets clear that the difference is not significant. The response time for the relative accounting is only 1.15% higher than those for the other approaches. Moreover, the standard deviation of the favor ratio (which will be large if some peers have favor ratios far from 1) is only 0.012 larger for the relative accounting than for the perfect and time-based schemes.

This setting illustrates that even when the time-based accounting performs the best (because all resources are identical), it is still possible to use the relative accounting without significant losses in the performance.

#### 4.5.3. Setting 3: Unfavorable Case (exponential distribution)

This setting has the same configuration as the previous, except that the heterogeneity in the size of tasks is modeled by an exponential distribution with mean 400, instead of a uniform one. That change is supposed to make the setting even worse for the relative accounting scheme, for which heterogeneous tasks may bring erroneous results to the calculation of the relative peer power.

Table 3 shows the response times and favor ratios for each type of accounting scheme. As expected, the perfect and the time-based accountings keep having very similar

performance. As for relative accounting, response time displays good results: its mean and standard deviation is only a little bit worse than before (with uniform distribution). Favor ratio, on the other hand, displays more pronounced unfavorable results. But note that the standard deviation is still quite small, indicating that fairness was not heavily affected. In our opinion, the behavior of relative accounting in this very unfavorable setting is still acceptable.

**Table 3. Response time and favor ratios for peers in setting 3**

	<b>Mean (Std Dev) of Response Time</b>	<b>Mean (Std Dev) of Favor Ratio</b>
<b>Perfect</b>	2,921.6 (2,783.6)	0.9524 (0.125)
<b>Relative</b>	2,951.2 (2,879.4)	0.9193 (0.175)
<b>Time</b>	2,922.1 (2,782.6)	0.9523 (0.125)

#### 4.5.4. Setting 4: Accounting Attack

This setting gauges the behavior of the accounting schemes when peers try to take advantage of the system by providing resources of poor quality. It is important that consumer peers are able to identify these resources. If providers can receive the same credit, no matter whether the resource is fast or slow, they have an incentive to provide the slowest possible ones, and the system collapses, a phenomenon known as the market for lemons [7].

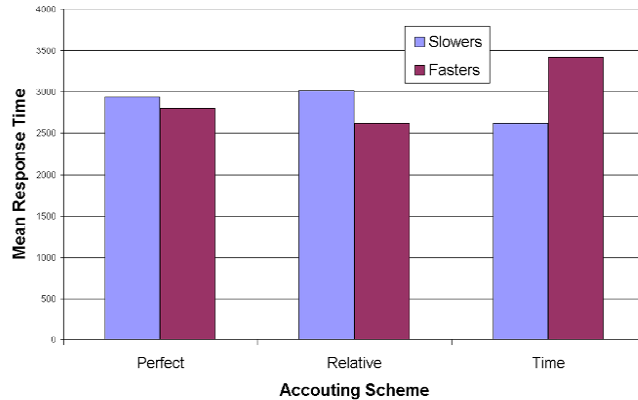
The setting is a grid with 20 peers, each with 25 machines whose power varies following the uniform distribution  $U(4,16)$ . However, 10 of such peers run 2 tasks on each machine, appearing for the grid to have 50 machines of power  $U(2,8)$ <sup>1</sup>. We name these peers slower, whereas the other 10 peers, which run 1 task per machine, are named faster. Jobs arrive every  $U(1,799)$  time units, each carrying 250 tasks, each task demanding  $U(200,600)$  time units to execute. Besides, in this setting, 1,000 jobs were processed by each peer.

As Figure 2 shows, when the perfect accounting is used, the slowest peers get no benefit from their behavior. In fact, as they take longer to process tasks, they are more prone to abort tasks from other peers. This makes them get less credit in the grid, which is reflected in their slightly worse mean response time when compared to the fastest peers.

---

<sup>1</sup> In practice this problem is exacerbated by the fact that the strategy followed by the malicious peers may introduce trashing in a resource when the tasks that share the resource cannot all fit in main memory, thus decreasing the power of the grid as a whole.

The relative accounting has similar behavior. However, the difference between the mean response time of the faster and slower peers is a little larger. The mean response time for the fastest peers is 7.13% shorter than when using the perfect accounting. The mean response time for the slowest peers, on the other hand, is 2.63% longer.



**Figure 2. Mean response time for setting 4**

For the time-based accounting, we see that slower peers get an advantage when compared to faster peers, creating an incentive for peers to adopt this strategy. The time-based approach gives two advantages to the slowest peers. First, it enables the malicious peer to run twice the number of tasks non-malicious peers can run simultaneously and thus to provide more favors to other peers. Second, it increases the execution time of each task, which benefits the malicious peer, because other peers value favors based on how long it takes to run their tasks.

**Table 4. Favor ratio for peers in setting 4**

	<b>Mean (Std Dev) of <i>fr</i> for slower peers</b>	<b>Mean (Std Dev) of <i>fr</i> for faster peers</b>	<b>Mean (Std Dev) of <i>fr</i> for all peers</b>
<b>Perfect</b>	0.931 (0.122)	0.976 (0.111)	0.953 (0.119)
<b>Relative</b>	0.911 (0.132)	0.989 (0.107)	0.950 (0.126)
<b>Time</b>	1.003 (0.117)	0.833 (0.117)	0.918 (0.145)

Looking at the favor ratios in Table 4 we see how the time-based accounting makes the grid prioritize the slowest peers. They get more for the resources they provide to the grid. Again, the relative accounting gets very close to the perfect one, with a standard deviation of favor ratio for all peers only 0.0077 larger than that for the perfect scheme.

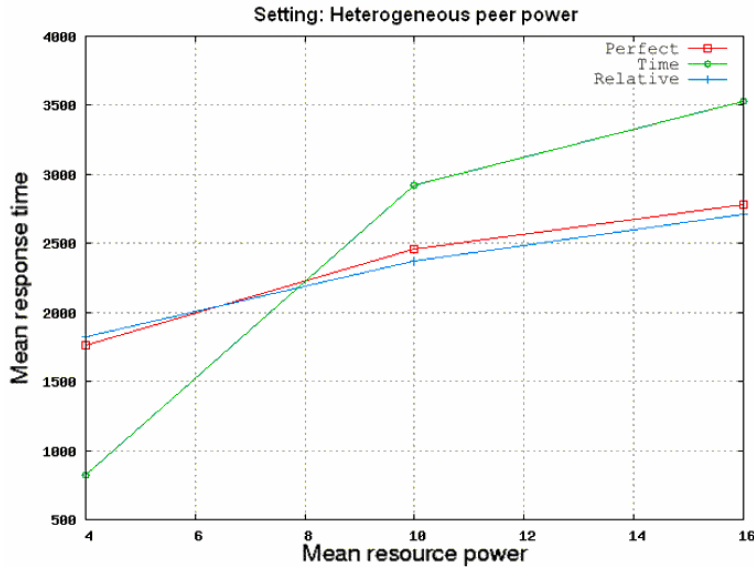
This setting shows that using the relative accounting makes the peer-to-peer grid robust against the problem of the market for lemons. It provides consumers with accurate enough information to prevent provider peers serving low-quality resources from gaining

an advantage over those providing high-quality ones. Although the relative accounting introduces a larger difference in the favor ratio mean between faster and slower peers than a perfect scheme, this difference (8.5%) is not significantly high. Since it is infeasible to deploy a perfect accounting in a peer-to-peer grid, we believe that the relative accounting is a good enough approximation. Moreover, relative accounting actually creates an incentive for peers to make available the best resources to the grid, what might be of overall interest.

#### *4.5.5. Setting 5: Heterogeneous peer power*

In this setting, we intended to evaluate the performance of the accounting schemes in a situation in which the total power of the peers in the grid is different. In order to accomplish that, we have fixed the number of resources per peer at 25 and varied their power values. Thus, we have created a grid comprised of 15 peers, of which 5 of them have total power 100, other 5 peers have total power 250, and the remaining 5 have total power 400. As the power of the peers is different, we have also submitted different workloads to each type of peer, in order to saturate them. More precisely, we have submitted 100, 250 and 400 jobs, each with 250 tasks, to peers of total power 100, 250 and 400, respectively. The size of the tasks was fixed at 400 time units. Besides, the mean inter-arrival time of jobs to the participant peers is obtained from the uniform distribution  $U(1,799)$ .

Figure 3 shows the mean response time of jobs for each type of peer. For perfect accounting, we would expect the response times to be independent in relation to the power of the peers. However, the simulations have shown that the mean response times increase with the peer power for all of the accounting schemes evaluated. We believe the reasons for that are twofold. First, peers with greater power obtain remote resources with lower power, thus running more tasks locally, and therefore benefiting less from remote resources. Second, since peers with lower power take longer to complete foreign tasks, such tasks are more likely to be aborted by local submissions.



**Figure 3. Mean response time for setting 5**

For perfect accounting, we have observed an increase of 58.1% in the mean response time for the peers of power 400 in relation to those of power 100. Relative accounting tracks this behavior very closely. However, by using the time-base accounting, the increase among the peers that have the slowest and the fastest resources is much higher (329.3%). This greater increase shows us that, by using a time-based accounting, low-power peers are benefited and receive much more resources from the grid than the high power peers.

**Table 5. Favor ratio of peers in setting 5**

	Mean (Std Dev) of favor ratio for peers of total power 100	Mean (Std Dev) of favor ratio for peers of total power 250	Mean (Std Dev) of favor ratio for peers of total power 400
<b>Perfect</b>	0.965 (0.052)	0.971 (0.080)	0.962 (0.102)
<b>Relative</b>	0.967 (0.044)	0.982 (0.065)	0.970 (0.085)
<b>Time</b>	1.124 (0.025)	0.922 (0.067)	0.797 (0.089)

It is also important to observe whether the accounting schemes make the resource exchanges fair for all peers, no matter the computing power they have. Table 5 shows the values of the favor ratios for each type of peer and accounting scheme. As we can see, by using perfect accounting, the ratio of consumed and donated resources is almost the same, no matter the power of the peers, i.e., changes in the peer power do not affect the dynamics of resource exchanging among the participant peers. A different behavior comes from the time-based accounting, in which there is a clear decrease in the favor

ratio as the total peer power increases. Those results show that time-based accounting benefits the peers with low computing power, given that the peers of power 100 have consumed much more foreign resources than they have donated to the grid. More precisely, that variation in the ratio is 29.2%. Meanwhile, the behavior of the favor ratio for the relative accounting is almost the same as for the perfect scheme: a little less than 1 for all types of peers.

## 5. RELATED WORK

The problem of accounting on grids has been considered before. The major difference between our solution and previous works is that we specifically target accounting among parties that do not trust each other, whereas other proposals assume that parties exchange correct measurement data.

In fact, in order to measure the resource utilization and power of resources, GridBank [9] and the DataGrid Accounting System (DGAS) [16] use resource meter agents deployed throughout the grid. We argue that deploying the functionality necessary for collecting utilization data faces serious trust issues for peer-to-peer grids. SweGrid Accounting System (SGAS) [14] uses only the time that the resources are available to run applications as accounting metric. In SGAS case, the accounting is simplified because all resources are equal. However, as we have shown in our results, for a more heterogeneous grid, using only time benefits providers of slow resources. In short, in GridBank, SGAS and DGAS the accounting is done in the provider side and sent to the consumer or a third-party bank. In a peer-to-peer grid, free-riding providers can send fraudulent accounting information, misleading the consumer. To circumvent this, we autonomously account the resource allocations, so there is no need of exchanging information among parties.

Furthermore, Thigpen et al. [27], DGAS and GridBank assume that there is a previous negotiation between the consumer and the provider before the resource utilization. To help the negotiation, they mention the need to estimate the cost for running a job prior to the use of the resource. This estimation could be supplied by the user or estimated using historical information. Instead, in our scheme, consumers and providers do not negotiate before execution and autonomously estimate the relative peer power during the job execution, what greatly simplifies the use of an accounting scheme from the user's point of view.

In fact, not requiring previous negotiation is a key feature of reciprocation-driven peer-to-peer resource-sharing systems [2] [4] [5] [6] [8] [13] [21]. These systems are built around the notion of *favor*, i.e. a resource access or content a peer provides to another. Different reciprocation-driven systems differ exactly on the strategy used to promote *fairness*, i.e. having a peer receive back from the system the same amount of favors it

made to other peers. This work aims to better evaluate the value of a favor, being it given or received. In principle, relative accounting could be used to refine existing reciprocation-driven systems, most of which assume (or are evaluated assuming) that all favors have equal value.

## **6. CONCLUSIONS**

We have analyzed three accounting schemes to support the implementation of an autonomous reputation-based allocation mechanism for peer-to-peer grids. The perfect scheme requires timely and accurate information on the power that all resources in the grid are able to deliver at any given time. It is very unlikely that such a scheme can be implemented in practical peer-to-peer grid systems. The time-based scheme, although very simple to implement and deploy, gives a noticeable benefit for peers with slower resources in detriment to those with faster resources. As a result, this accounting scheme is vulnerable to a simple attack in which malicious peers voluntarily reduce the power of their resources by executing more than one task at a time in each resource.

The proposed relative accounting behaves closer to the perfect accounting and is also very simple to implement and deploy. Even for the less favorable (unlikely in practice) setting in which the computing resources that form the grid are homogeneous and the workload submitted to the grid is heterogeneous (see Section 4.5), our experiments have shown that the relative accounting is only slightly worse than the perfect accounting.

The main difference when comparing relative accounting against the perfect accounting is that it is slightly biased towards peers with faster resources. We believe, however, that this will not have a negative impact on the grid. On the contrary, it makes in the best interest of the peers to provide their most powerful resources to the grid. Peers therefore have an incentive to increase their local power and as a consequence the power of the grid as a whole.

We are currently implementing relative accounting in OurGrid. It should be available in the upcoming 4.0 release of OurGrid. As future work, we also intend to implement the multiple service economy briefly discussed in Section 4.1 (and fully discussed in [20]), considering processing and data transferences as the available services.

## **7. ACKNOWLEDGEMENTS**

This work was partially developed in collaboration with HP Brazil R&D. Francisco Brasileiro and Walfredo Cirne would like to thank the financial support from CNPq/Brazil (grants 300.646/96 and 302.317/03, respectively). Thanks also to Katia Saikoski and to the reviewers for the comments and suggestions.

## 8. REFERENCES

- [1] E. Adar and B. Huberman. "Free riding in Gnutella". *First Monday – Peer-reviewed Journal on the Internet* 5(10), October 2000. Available online at [http://www.firstmonday.dk/issues/issue5\\_10/adar/](http://www.firstmonday.dk/issues/issue5_10/adar/). Last access 01/31/2006.
- [2] E. Anceaume, M. Gradinariu, A. Ravoaja. "Incentive for P2P fair resource sharing". IEEE P2P Conference, September 2005.
- [3] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg. "OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing". *9th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003
- [4] N. Andrade, M. Mowbray, W. Cirne and F. Brasileiro. "When Can an Autonomous Reputation Scheme Discourage Free-riding in a Peer-to-Peer System?" *Proceedings of the 4th Workshop on Global and Peer-to-Peer Computing (GP2PC)*, USA, April 2004.
- [5] N. Andrade, F. Brasileiro, W. Cirne and M. Mowbray. "Discouraging Free-riding on a Peer-to-Peer Grid". *Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC'13)*, June 2004.
- [6] N. Andrade, F. Brasileiro, W. Cirne and M. Mowbray. "Automatic Grid Assembly by Promoting Collaboration in Peer-to-Peer Grids". Submitted for Publication. HP Labs Technical Report HPL-2005-22, 2005.
- [7] G. Akerlof. "The Market for Lemons: Quality Uncertainty and the Market Mechanism". *Quarterly Journal of Economics* 84 (3), 488-500, 1970.
- [8] D. Banerjee, S. Saha, S. Sen and P. Dasgupta. "Reciprocal resource sharing in P2P environments". *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05)*, pp. 853-859, ACM Press, 2005.
- [9] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration". *17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003)*, IEEE Computer Society Press, USA, April 22-26, 2003, Nice, France February 2003.
- [10] A. Butt, R. Zhang, Y. Hu, "A Self-Organizing Flock of Condors". *SuperComputing 2003*, November 2003.
- [11] F. Cappello, S. Djilalia, G. Fedak, T. Heraulta, F. Magniettea, V. Nérib and O. Lodygenskyc. "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid". *Future Generation Computer Systems*, 21(3):417-437, March 2005.
- [12] W. Cirne, F. Brasileiro, N. Andrade, R. Santos, A. Andrade, R. Novaes, M. Mowbray. "Labs of the World, Unite!!!". *Technical Report UFCG/DSC 07/2005*. Available online at <http://www.ourgrid.org/twiki->

public/pub/OG/OurPublications/Labs\_of\_the\_ World\_Unite\_v12.pdf. Last access 01/31/2005.

- [13] B. Cohem. "Incentives Build Robustness in BitTorrent". *1st Workshop on Economics of Peer-to-Peer Systems*, SIMS Berkeley, June 2003.
- [14] E. Elmroth, P. Gardfjell, O. Mulmo, and T. Sandholm. "An OGSA-Based Bank Service for Grid Accounting Systems". In J. Wasniewski et al. (eds). *Lecture Notes in Computer Science: Applied Parallel Computing. State-of-the-art in Scientific Computing*. Springer Verlag, 2004.
- [15] I. Foster and A. Iamnitchi. "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing". *Second International Workshop on Peer-to-Peer Systems, (IPTPS'03)*, February 2003, Berkeley, CA.
- [16] A. Guarise, R. Piro, A. Werbrouck. "An Economy-based Accounting Infrastructure for the DataGrid". *4th International Workshop on Grid Computing (GRID2003)*, Phoenix, Arizona (USA), November 17, 2003.
- [17] D. Hughes, G. Coulson and J. Walkerdine. "Free Riding on Gnutella Revisited: The Bell Tolls?" *IEEE Distributed Systems Online*, Volume 6, Issue 6, June 2005.
- [18] Y. S. Kee, H. Casanova, A. Chien. "Realistic Modeling and Synthesis of Resources for Computational Grids". *SuperComputing 2004*, Pittsburgh, PA, November 2004.
- [19] V. Lo, D. Zhou, D. Zappala, Y. Liu, and S. Zhao. "Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet". *Third International Workshop on Peer-to-Peer Systems, IPTPS 2004*.
- [20] M. Mowbray, F. Brasileiro, N. Andrade, J. Santana, W. Cirne. "A Reciprocation-Based Economy for Multiple Services in Peer-to-Peer Grids". UFCG/DSC Tech Rep 01/2006.
- [21] N. Ntarmos and P. Triantafillou, "SeAI: Managing Accesses and Data in Peer-to-Peer Data Sharing Networks", *4th IEEE International Conference in Peer-to-Peer Computing (P2P2004)*, August 2004.
- [22] OurGrid Web Site. <http://www.ourgrid.org>. Last access 01/31/2006.
- [23] R. Santos, A. Andrade, W. Cirne, F. Brasileiro, N. Andrade. "Accurate Autonomous Accounting in Peer-to-Peer Grids". *3rd Workshop on Middleware for Grid Computing (MGC2005)*, Nov. 2005.
- [24] L. Sarmenta. "Sabotage-tolerance mechanisms for volunteer computing systems". *Future Generation Computer Systems*, 18(4):561--572, 2002.
- [25] S. Saroiu, P. Gummadi and S. Gribble. "A Measurement Study of Peer-to-Peer File Sharing Systems". *Multimedia Computing and Networking 2002 (MMCN'02)*, USA, January 2002.

- [26] K. Shudo, Y. Tanaka and S. Sekiguchi. "P3: P2P-based Middleware Enabling Transfer and Aggregation of Computational Resources". *Proceedings of 5th International Workshop on Global and P2P Computing (GP2PC)*, May 2005.
- [27] W. Thigpen, T. Hacker, B. Athey and L. McGinnis. "Distributed Accounting on the Grid". *Proceedings of the 6th Joint Conference on Information Sciences (JCIS)*, 2002.