# Enforcing Fairness in Asynchronous Collaborative Environments

**Guillaume Pierre, Maarten van Steen**
*VU University Amsterdam, The Netherlands*

## ABSTRACT

Many large-scale distributed applications rely on collaboration, where unrelated users or organizations share their resources for everyone's benefit. However, in such environments any node may attempt to maximize its own benefit by exploiting other's resources without contributing back. Collaborative systems must therefore deploy strategies to fight free-riders, and enforce collaborative behavior. This chapter explores a family of mechanisms to enforce fairness in asynchronous collaborative environments, where simple tit-for-tat policies cannot be used. Our solutions rely on enforced neighborhood relations, where each node is restricted in the choice of other nodes to collaborate with. This creates long-term collaboration relationships, where each node must behave well with its neighbors if it wants to be able to use their resources.

## INTRODUCTION

Service-Oriented Architectures offer the vision of distributed applications offering functionality to each other, such that complex applications may be realized mostly by composition of existing software components provided by independent parties. An example is the recent popularity of Web mashups, where any programmer can take advantage of service-oriented functionality offered through Web services. However, users of service-oriented applications do expect reasonable performance. This requirement can be translated into: how can a service provide constant performance regardless of the request load addressed to it by independent third parties? Obviously, a single server machine cannot handle arbitrary amounts of load so one must design services such that they can expand their capacity by using additional computing resources when necessary.

One way a service may obtain temporary access to extra resources when it needs them is the use of collaborative environments. Such environments are characterized by multiple users sharing their resources for everyone's benefit. For example, peer-to-peer file sharing applications can improve everyone's download speed of a file under the condition that those users are willing to donate their resources to upload file contents (Cohen, 2003). Similarly, a service operator may use a collaborative content delivery network, which relies on the willingness of Web server administrators to help each other if one experiences a temporary overload (Pierre and van Steen, 2006); another possible method is to use grid computing, where system administrators are willing

to contribute their resources in exchange for future use of global resources (Foster and Kesselman, 1998).

One important issue in such environments is free riding, where some users try to use the shared resources without contributing an equivalent quantity of resources back to the system (Adar and Huberman, 2000). Free riding can be extremely detrimental to the performance of collaborative systems as it decreases the quantity of resources available to users as a whole. Additionally, it produces strain on the remaining good nodes in the system, which reduces the incentive to contribute positively.

An efficient mechanism to enforce collaboration is the tit-for-tat policy, as implemented for example in the BitTorrent file sharing system (Cohen, 2003). This policy dictates that, after a first altruistic interaction, resources are granted to a user under the condition that an equivalent amount of resources is simultaneously contributed back. A few properties of BitTorrent make this scheme effective and easy to apply. First, collaboration is symmetric, meaning that collaboration happens pairwise with no third party involved. Fairness enforcement can thus be realized by the two concerned peers themselves, without requiring the need for external services such as reputation systems. Second, collaboration is local in time in that the balance of respective contributions must be judged as fair by both parties at any instant of the collaboration. The system therefore does not need to maintain a memory of past interactions.

However, tit-for-tat is not a panacea for solving all fairness issues in collaborative environments. Many such environments rely on asynchronous collaboration, where the services are not provided simultaneously from A to B and from B to A. A good example is a collaborative content distribution network, where Web servers call each other for help only when they experience an overload. For such environments we need more sophisticated mechanisms.

This chapter explores a family of mechanisms to enforce fairness in asynchronous collaborative environments, based on observations from Axelrod (Axelrod, 1984). These observations state that cooperation can emerge only when:

- Nodes retain a unique identity over time
- Interactions are repeated many times between the same pairs of nodes

The intuition between these rules is that, to sustain collaboration between the well-behaving members (and to exclude free-riders), one must rely on some memory of past interactions with other nodes. In a large-scale system this implies that a given node regularly interacts with the same partners over and over again, to have the ability to gain some confidence that they will behave well in the future.

The following sections explore the application of these general principles to two classes of asynchronous collaborative environments. First, we study fairness enforcement in a collaborative content distribution network. In such a system, Web servers may request each other's help when they experience an overload. The interaction is thus necessarily asynchronous because a currently overloaded server cannot be of much help to another overloaded server. It makes more sense that underloaded servers help overloaded servers, thereby creating asynchronous collaboration. We

then turn to peer-to-peer grids, where a user can allow jobs from other users to execute on his/her resources in order to gain the rights to later run jobs at other sites. Here, in addition to the fundamental asynchronous nature of the system we need to deal with an extra difficulty: restricting the number of neighbors where a node can execute its jobs has an unacceptable performance impact on the Grid as a whole.. We therefore focus on the way to enforce repeated interaction without limiting the scope of machines where a user may submit jobs.

## WHAT IS FAIRNESS?

Defining fairness in collaborative environments is equally difficult as it is in human societies. Ideally, fairness would mean that each party receives resources proportional to their own contribution to the collaborative system. For example, Jain defines fairness in the context of a single resource that must be shared fairly between multiple users (Jain et al., 1984). This metric is meant to evaluate instantaneous fairness, in the sense that one user of the shared resource is not unfairly discriminated compared to the others. However, characterizing fairness in asynchronous collaborative systems is much more complex. On the one hand, a good system must allow resource sharing to be unfair at any point of time: certain nodes only contribute resources while others only receive them. On the other hand, it must distinguish free-riders who intentionally refuse to contribute sufficient resources to the system from nodes that are willing to share their resources but are genuinly unable to do so (e.g., a collaborative Web server may need to refuse to help another one if it already experiences a high load).

In this chapter, we use a definition of fairness similar to that of (Böhm and Buchmann, 2009): the goal of a fairness enforcement system is that the presence of free-riders does not significantly impact the benefits that collaborative nodes get from collaboration. Note that this largely implies that the free-riders do not draw any significant benefit from their actions.

## FAIRNESS IN A COLLABORATIVE CONTENT DISTRIBUTION NETWORK

A collaborative content distribution network is an overlay of independent Web servers that organize to offload requests from each other when a server is overloaded (Lal, 2007, Pierre and van Steen, 2006). This guarantee is important because a Web server should expect its request traffic to contain significant bursts of activity (Crovella and Bestavros, 1996). Dimensioning any Web server according to the greatest expected load peak can be extremely expensive, so in practice very few servers are sufficiently provisioned to sustain, for example, the load peak created by having a major news site publish a link to the concerned Web server (Adler). In such situations, it becomes interesting to provision a server for its average load only, and rely on other currently underutilized servers to serve the excess load that the server will occasionally be receiving.

For such a system to work in practice, each server must have the assurance that when an overload will occur, another underloaded server will accept to serve as a backup. A free-riding strategy in such a system could consist of utilizing other server's resources during periods of overload, but deny other servers to use the local resources when the server is underutilized. This system clearly

belongs to the category of asynchronous collaborative systems: one server can help another one only if it has spare capacity to donate. On the other hand, although collaboration will at any instant be only unidirectional, no server wants to be systematically exploited by its peers so a fair balance of respective contributions should be maintained in the long run.

## Fixed Neighborhoods

The first issue to address is that, like any collaborative environment, collaborative content distribution networks must allow certain levels of altruism to initiate the collaboration. One server must first accept to donate resources to another server with no immediate counterpart, in the hope that the favor will be returned later on. However, although altruism is a necessity to allow collaboration to start, we do not want to allow free-riders to exploit the generosity of every node in the system without ever reciprocating the favor.

We solve this problem by statically assigning a fixed set of helpers to each server. This means in practice that an external entity defines a set of neighbors for each server. These neighboring links are symmetric, so it is easy for a server to decide if a request for help comes from a server that is authorized to do so. This also enforces repeated interaction, since a given server is forced to establish long-term relationships with its neighbors. A free-riding server which would exploit the altruism of its designated neighbors without reciprocating the help would quickly be excluded by these neighbors, and therefore become unable to use the collaborative system any more.

## Asynchronous tit-for-tat

In an asynchronous collaborative environment such as a collaborative content distribution network, it is impossible to maintain a perfect balance of mutual contributions at all times. Consider a pair of collaborative servers A and B. One of the two (say, A) has to donate resources to B, before B can reciprocate later on. On the other hand, A cannot have hard guarantees that B will effectively reciprocate when requested to do so. The only guarantee that can be provided here is trust, based on previous experience among these two particular servers.

Such trust can be built using two simple mechanisms. First, each server must maintain the balance of mutual contributions it had with each of its neighbors. This allows to limit the amount of generosity toward any neighbor, and to deny resources to a neighbor which would request resources without sufficiently reciprocating in the long term. Second, the maximum tolerated imbalance in respective contributions can be adjusted according to past experience. At the start of a collaboration, this maximum imbalance can be set to a relatively low value, so each server minimizes the risk of being exploited by the other. Later on, if the neighbor behaves well and reciprocates the collaboration, then the maximum imbalance may be increased gradually to expand the scope of the collaboration.

More formally, each server maintains three variables for each of its neighbors: *TotalBytesConsumed*, *TotalBytesServed* and *DeficitThreshold*, representing respectively the resources consumed from and donated to the neighbor, and the maximum tolerated imbalance for this neighbor. A request for resources originating from this peer will be granted under two

conditions: (i) the requested server is not itself currently experiencing overload; and (ii) *TotalBytesConsumed - TotalBytesServed* < *DeficitThreshold*.

While this simple model accurately represents the quantity of resources donated by each server to another, it is not sufficient to account for the quality of service with which one server has helped another. In a content delivery network, it matters a lot at which data transfer rate a server has served its neighbor's data to external clients. Thus, contributions should be expressed not only in terms of the quantity of donated resources, but also with their quality of service. However, the rate at which a neighbor can serve any particular connection does not only depend on the resources it is capable or willing to contribute. The location of the client and the characteristics of the Internet path to that client also contribute to determine the transfer rate of this connection. We should therefore not punish a neighbor server for slow data rates observed on any particular connection. On the other hand, one can aggregate all requests served by a particular neighbor (for example, using exponential weighted moving average functions) and derive trends to indicate if the global quality of service offered indicates artificial contention created by the neighbor server. By comparing the trend of connection rates delivered by the neighbor to the one it provides to its own clients, a server can decide if the neighbor is playing fair or not, and consequently reward or punish it.

## Validating claims

Enforcing collaboration in a collaborative content distribution network creates one extra difficulty that is not found often in other collaborative systems: to be able to suitably reward or punish a neighbor server according to its behavior, a server must be able to verify the claims that a neighbor server has actually served so many requests at so much data rate. However, the nature of collaborative Web sites makes such verification hard to realize: after a server has redirected a client request to one of its neighbors, it is no longer involved in this request. It therefore cannot verify whether the request was actually served, and at what rate.

One simple solution consists of involving (a fraction of) the clients, such that they report the quality of service they experienced to the origin server. Such systems have actually been built, for example to allow one to detect whether fraudulent content was delivered to the user (Popescu et al., 2003). However, this requires that a significant fraction of end users accept to install extra software to issue such reports.

A possible alternative solution is to exploit the properties of a new mechanism for request redirection, named Versatile Anycast (Szymaniak et al., 2007). With versatile anycast, one server can handoff any of its TCP connections with clients to be served by a neighboring server. The handoff is realized at the IP layer, so that the client-side application does not notice it is being redirected. After redirection, the connection traffic is routed directly between the client and the neighboring server, with no traffic indirection through the origin server. This form of redirection has the advantage that it allows the origin server to check how much data its neighbors actually deliver to clients, and at which rate. In this scheme, a new client request is always opened with the origin server. If the origin server is overloaded, it can then hand off some of its connections to its neighbors. The neighbor is asked to serve the client requests, then to hand off the connection back to the origin server before closing it. The origin server can then check the data offsets of the

TCP socket, and verify how much data has actually been exchanged with the client. The origin server can also measure how much time has elapsed between the connection was handed off from the origin to its neighbor and the time when it was handed off back, and derive the average data rate with which this connection has been served.

## Evaluation

We evaluated the proposed approach by studying an implementation of two neighboring servers connected to the Internet by emulated ADSL-like connections, each with 2 Mb/s download bandwidth and 240 kb/s upload bandwidth. We assume that the documents to be served are available at both servers, so no document replication cost is incurred in these experiments. Each server is addressed by a bursty workload, generated using a modified version of S-Client (Banga and Druschel, 1997). This tool allows us to control the burstiness of the traffic addressed to the servers. We express burstiness according to the notation in (Menasce and Almeida, 2001), using two parameters *(a, b)*.

Parameter *a* denotes the ratio between the maximum observed request rate and the average rate during the evaluation period. Parameter *b* denotes the fraction of time during which the instantaneous request rate exceeds the average request rate.

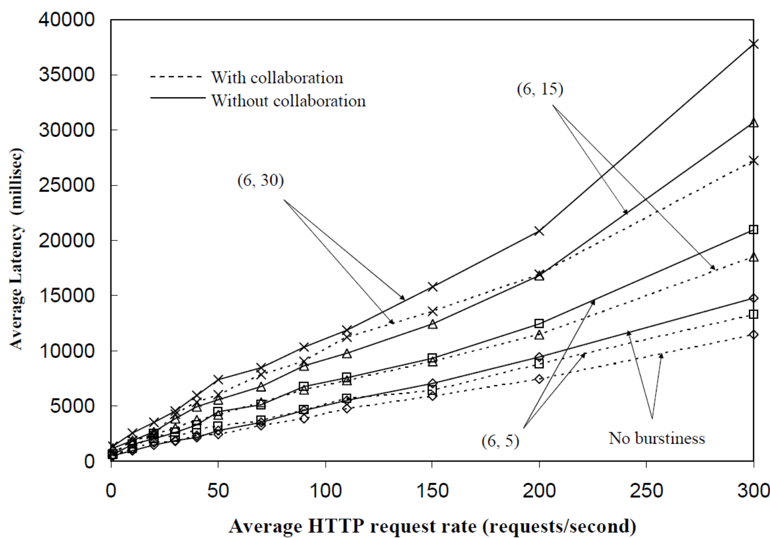



*Figure 1: Performance of two Web servers with and without collaboration*

Figure 1 shows the average client-perceived latency of requests addressed to the two servers for several levels of traffic burstiness, with and without collaboration between the two servers. We observe three phenomena: first, obviously, request latency grows when the average request rate grows. Second, for the same average request rate, a bursty traffic is more difficult to serve, as indicated by greater request latency. Finally, we observe that collaboration between the two servers allows to significantly reduce request latencies, especially when the traffic is more bursty. Two servers receiving similarly bursty traffic therefore have a common interest to collaborate.

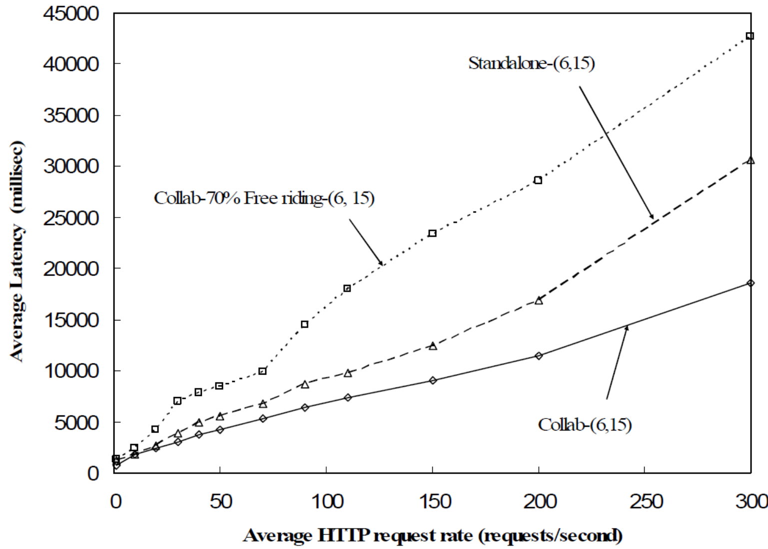



*Figure 2: Performance of a Web server with a free-riding neighbor*

Figure 2 shows the damage that a free-riding neighbor can create. In this example, the free-riding neighbor denies 70% of requests for help from the origin server. In such cases, the origin server must first incur the latency of the denied request for help, then the cost of serving the request itself. We observe that the free-riding neighbor causes an increase of request latency compared to the scenario where the neighbor collaborates, but also compared to the standalone case with no collaboration. We conclude that the best way to behave with a free-riding neighbor is not to collaborate with it any more, since doing so creates an extra burden compared to no collaboration.

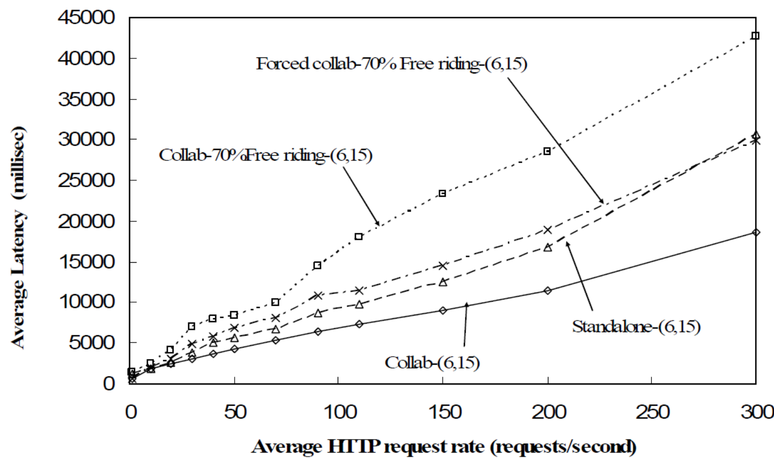



*Figure 3: Performance of a Web server with a free-riding neighbor, and enforced collaboration*

Figure 3 shows the performance of the same server when the enforced collaboration mechanisms are activated. The average request latency in the presence of a free-riding neighbor becomes close to the case with no collaboration. This demonstrates that the free-riding neighbor no longer impedes the performance of the origin server.

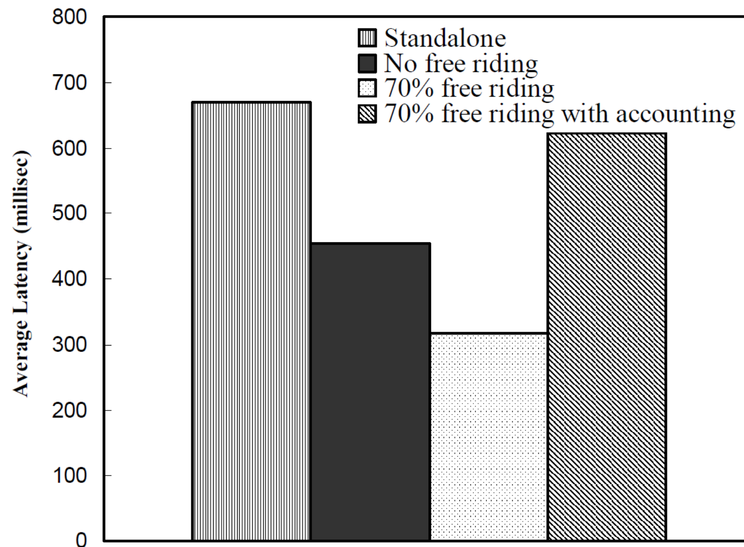



*Figure 4: Performance of a free-riding server*

We conclude this evaluation section with Figure 4, which shows the performance experienced by the free-riding server in these different scenarios. We see that free-riding with no accounting gives this server an extra performance advantage compared to regular collaboration with no free-riding: the free-riding server can exploit the resources of its collaborative neighbor, without reciprocating. The request latency of the free-riding server therefore improves. However, when we enable the accounting mechanism to enforce collaboration, the free-riding server sees its performance drop to a value close to the case with no collaboration at all. A server therefore has no incentive any more to free ride: using the enforced collaboration, the optimal strategy of any server is to collaborate with its neighbors, for everyone's shared benefit.

## FAIRNESS IN A PEER-TO-PEER GRID

A different kind of asynchronous collaborative environment is a peer-to-peer grid (Foster and Kesselman, 1998, Weel, 2008). Here, collaboration happens when a node issues a computational job to be executed by some number of its peers. Similar to the situation of collaborative content delivery networks, each compute node is expected to accept running computations on behalf of other users, in order to gain the rights to later execute jobs on other nodes. Peer-to-peer grids however present two important differences with collaborative content distribution networks. First, in a peer-to-peer grid, a node expecting to run a job at other nodes expects to find several remote peers simultaneously available to run the computation. Second, it is not a necessity that jobs should start executing immediately after they are requested: a reasonable delay before execution starts is acceptable in most cases.

In such a system, restricting collaboration to a fixed number of neighbors severely reduces the performance of the Grid itself: if this set is significantly smaller than the total size of the grid, this greatly restricts the available periods when a sufficient number of neighbor nodes can be simultaneously available1. Increasing the number of neighbors per node is not a satisfactory solution either: doing this would increase the possibility for a node to free ride, by exploiting the initial altruism of each of its neighbors.

We therefore see that a node cannot be restricted to running jobs only at its direct neighbors without greatly reducing the efficiency of the system. In the worst case, a computation request would require more nodes than the number of neighbors, and thus could not execute at all. On the other hand, we need to keep a notion of neighbors, since these are the core of the mechanisms to avoid exploitation of the well-behaving nodes.

We address this dilemma by building an economic system between nodes: each node is assigned a small number of neighbors. A node can of course trade resources with its direct neighbors, similarly to the solution discussed for collaborative content distribution networks. However, it can also trade resources with nodes that are not its direct neighbors: in such case, it must find a path of neighboring nodes that leads to the desired compute node. Intermediate nodes in this path are thus expected to mediate the collaboration between the requester node and the compute node. For example, if node A has a credit with node B, and node B has a credit with node C, then we can use the path A → B → C: B can act as a mediator to allow node A to use resources from node C. The problem thus translates into being able to find a path of nodes having credit with each other from the requesting node to the providing nodes (we assume that each compute node can carry only one computation at a time).

## Fairness enforcement algorithm

Fairness enforcement in our collaborative grid works in two phases. First, when a job scheduling request is issued by one node requesting resources from its peers, the grid uses a decentralized scheduling algorithm to identify nodes that will be simultaneously available in the near future, and may be used to run this job (Fiscato et al., 2008). This algorithm initially selects a (random) set of nodes capable of executing the job together. It then iteratively improves its choice of nodes to find groups capable of starting the execution as soon as possible. This algorithm identifies groups of nodes based on their scheduled availability only, irrespective of fairness issues.

Each time an improved schedule is found for a given job, the initiator must check if it will be able to acquire rights to use the selected nodes. In other terms, it must find a path in the graph of neighboring relationships that links it to each of the selected nodes, and where each intermediate node is willing to mediate the resource usage. If at least one such path cannot be found, the schedule is declared invalid. The initiator then requests the scheduling algorithm to propose another solution, and so on.

A path of neighboring nodes will successfully mediate access to resources under the condition that each node would normally accept the job if it was initiated by its direct predecessor in the path. For example, a request from node A for a 1-hour long job to be executed by node C will be successful using path A → B → C if A has sufficient credit with B, and B has sufficient credit with C. If the path is found to be valid, then B will spend 1 hour of its credit with C, and gain 1 hour of credit with A. One could imagine a variant where node B would charge an additional contribution to A in payment for its brokering service.

Fairness enforcement within the whole collaborative grid is realized by the conjunction of multiple local neighboring relationships. In the example above, if A is a free-rider, then after

exploiting B's initial generosity, A will not be able to use any path A → B → *. After it has exhausted the generosity of all its neighbors, A is effectively excluded from the collaboration, unless it starts reciprocating again.

One important case in this scheme is the case where A and C behave well in the system, but the chosen mediator B is a free-rider. In such case, C will deny the resources to B so the path is unusable. A must therefore be able to find an alternative path composed of well-behaving nodes that connects it to C.

## Finding paths in the overlay

In the above described system, it is crucial to design the graph of neighboring relationships carefully. This graph should have the following properties:

1. Nodes should be able to efficiently find paths from each other in the graph.
2. If one path fails, then nodes should be able to find an alternative path to the same destination.
3. Neighboring relationships must be symmetric: if node A considers B as its neighbor, and potentially requests B for resources, then B must also consider A as its neighbor and regularly request A for resources so that A can balance its credit with B.

The first requirement suggests the use of a DHT overlay between the compute nodes. Each node is initially assigned an ID by an external entity. When it joins the DHT, its list of neighbors in the fairness system is defined as the list of its fingers in the DHT. Organizing nodes along a DHT structure allows nodes to find paths of length *O(log n)* from each other, where n is the number of nodes in the system.

Most existing DHTs also allow to support the second requirement: if one path is considered as invalid, then one can easily find alternative paths leading to the same destination. Supporting the third requirement is however more difficult: most DHT systems do not impose that the links between their nodes are symmetric. This rules out traditional DHTs like Chord and Pastry.

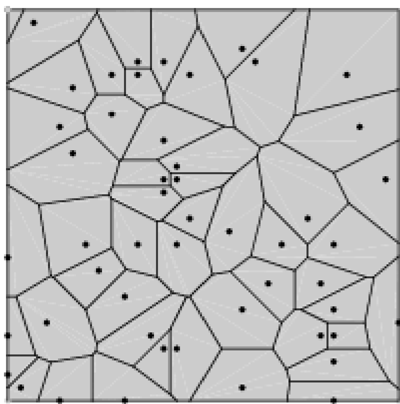


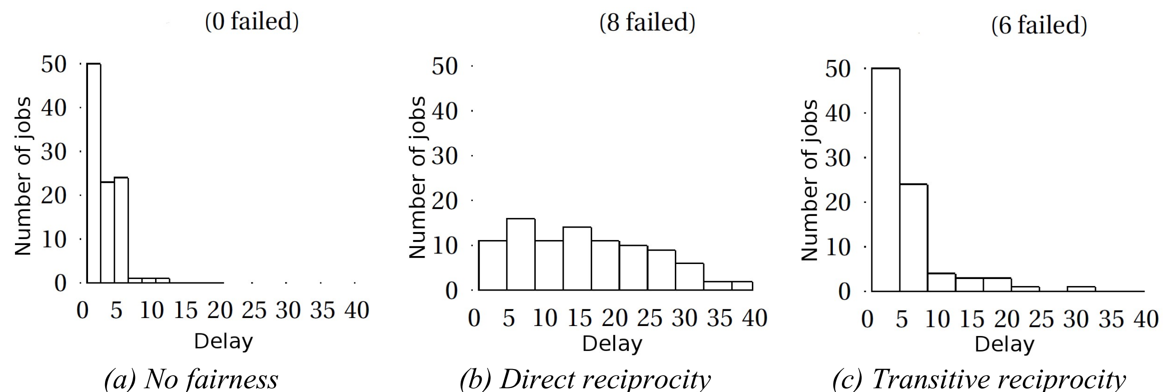
*Figure 5: A Voronoi diagram (Weisstein)*

We decided to base our neighboring overlay on Voronoi Diagrams, as is also done in the VoroNet overlay (Beaumont et al., 2007). In this overlay, each node is assigned a 2-dimensional coordinate randomly by an external entity. A Voronoi diagram associates each node s with a cell C(s), consisting of all points closer to s than to any other node. An example Voronoi diagram is shown in Figure 5. Each node, represented by a dot, maintains neighborhood relationships with the nodes responsible for contiguous cells. A Voronoi-based overlay has several interesting properties: (i) one can easily route messages by using geographical routing, where a message is always forwarded to one's neighbor that is closest to the destination; (ii) geographical routing also makes it easy to find alternative paths to a destination, for example by routing a message to one's second-closest neighbor; (iii) neighborhood relationships are symmetric by construction; (iv) Voronoi diagrams guarantee short paths between nodes, and maintain a small node degree, in average around 6.

When a node joins the Grid, it is assigned a coordinate by an external entity. It then joins the VoroNet overlay, by routing a message to its new coordinate and establishing neighborhood relationships with the nodes holding a cell contiguous to its own. On average this neighborhood will contain 6 links; the new node is thus forced to establish good reciprocating relationships with them in order to benefit from the grid after using the initial altruism that the new neighbors will grant. Neighbors can easily check that the new node should indeed be included in their list. Importantly, when new nodes join or old nodes leave the system, the implied changes in neighborhood relationships remain local to the coordinates of the joined or departed node, so most neighborhood relationships remain unchanged.

## Evaluation

We evaluate the performance of the fairness enforcement algorithm on a simulated 100-node grid. Every time unit, we submit 20 jobs requesting 5 machines for a duration of 5 time units each. Jobs are created during the first 5 time units, then no more job is submitted until the grid becomes idle again. Jobs are issued at randomly selected nodes. Each experiment lasts 100 time units: a job that has not started executing after this delay is considered as having failed.

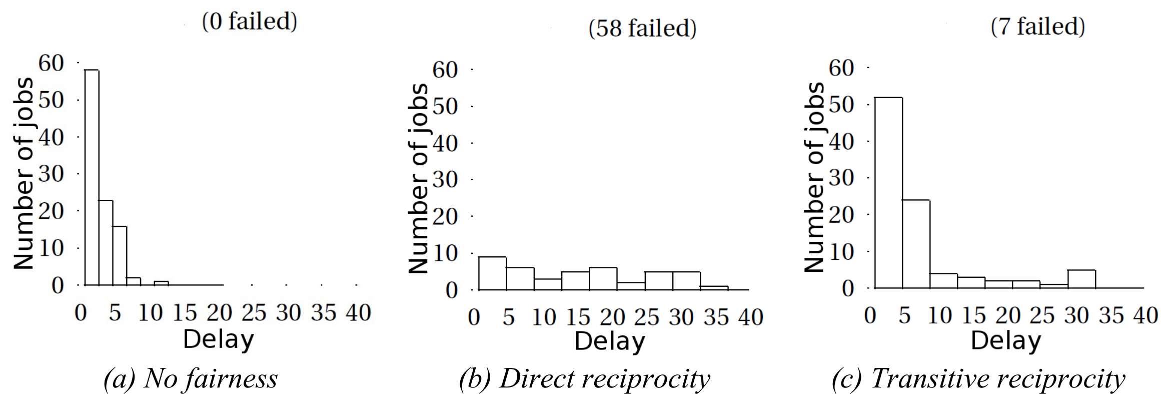


*(a) No fairness* *(b) Direct reciprocity* *(c) Transitive reciprocity*

*Figure 6: Impact of the fairness algorithms on scheduling performance*

Figure 6 shows the impact of the fairness algorithm on the scheduling algorithm, when no free-rider is present. When no fairness algorithm is present, most jobs start executing within 6 time

units after they are submitted. No job fails. When using direct reciprocity, nodes exchange resources only with their direct neighbors, without actually building paths to nodes further away in the overlay. The scheduling quality drops considerably: the waiting time before a job starts increases greatly, and 8 jobs fail. When using transitive reciprocity, nodes are authorized to build paths of any length to each other through the overlay, and trade resources along these paths. The quality of scheduling is nearly as good as the first case, with most jobs starting to execute within 10 time units after being submitted. We however note that 6 jobs fail, meaning that a number of paths are considered invalid. Although in this experiment no node is actively free-riding, not every node has the opportunity to execute enough jobs for its peers before it needs resources from its neighbors.




*(a) No fairness* *(b) Direct reciprocity* *(c) Transitive reciprocity*

*Figure 7: Performance in the presence of 10% free-riding nodes*

Figure 7 shows the impact of the presence of 10% free-riding-nodes in the system. When no reciprocity is present, jobs get scheduled similarly to the first case, albeit with slightly greater delays. In other terms, free-riders can exploit the resources of the well-behaving nodes. Direct reciprocity enforces fairness, but at an unacceptably high cost: 58 jobs fail, most of which were issued by well-behaving nodes. When using transitive reciprocity, we enforce fairness at a much more acceptable cost. The delay before jobs can execute is higher than with no reciprocity, due to the higher number of failing paths passing through a free-rider. However, only 7 jobs fail: most of these jobs are the ones issued by free-riders after their neighbor's initial generosity has expired.

## CONCLUSION

Many large-scale distributed applications rely on collaboration, where unrelated users or organizations share their resources for everyone's benefit. However, in such environments any node may attempt to maximize its own benefit by exploiting other's resources without contributing back. Collaborative systems must therefore deploy strategies to fight free-riders, and enforce collaborative behavior.

Tit-for-tat mechanisms have been deployed successfully in systems such as BitTorrent. However, tit-for-tat requires that collaboration is local in time. We studied two examples of asynchronous collaborative systems, where collaboration cannot be local in time. In such systems, applying instantaneous tit-for-tat mechanisms is not sufficient because one node must help the other before the second can reciprocate. Nodes therefore need to maintain a memory of past interactions, and

determine their collaborative behavior according to this past experience. This requires that each node, including the well-behaving ones, is restricted to a limited number of other nodes to interact with. In the case of collaborative content distribution networks this is relatively easy to achieve, as in principle any node may help any other when the need arises. We can thus build efficient pairwise fairness enforcement mechanisms. In the case of a collaborative grid, the application performance depends on the ability to use any peer in the system to execute jobs. Direct pairwise reciprocity mechanisms severely hamper the performance of the system itself. Instead, we have shown that using simple transitive economic incentives, one can enforce collaboration with reduced impact on the overall system performance.

Incentive mechanisms discussed in this chapter have the property that they rely only on local decisions: each node decides autonomously on what is fair or not, without the need to report selfish behavior to a higher authority. Yet, the ability of free-riders to disrupt the collaboration is restricted to the initial generosity of their direct neighbors. We believe that this property is key to building truly large-scale collaborative systems, where unrelated nodes gracefully collaborate with each other, whether they like it or not.

## REFERENCES


(Adar and Huberman, 2000) Adar, E. and Huberman, B. A. (2000). Free riding on Gnutella. First Monday, 5(10).

(Adler) Adler, S. The slashdot effect: An analysis of three internet publications. http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html.

(Axelrod, 1984) Axelrod, R. (1984). The Evolution of Cooperation. New York: Basic Books.

(Banga and Druschel, 1997) Banga, G. and Druschel, P. (1997). Measuring the capacity of a web server. In Proceedings of the USENIX Symposium on Internet Technologies and Systems.

(Beaumont et al., 2007) Beaumont, O., Kermarrec, A.-M., Marchal, L., and Riviere, E. (2007). VoroNet: A scalable object network based on Voronoi tessellations. In Proceedings of the Parallel and Distributed Processing Symposium.

(Böhm and Buchmann, 2009) Böhm, K., and Buchman, E. (2009) Free Riding-Aware Forwarding in Content-Addressable Networks. To appear in the VLDB Journal.

(Cohen, 2003) Cohen, B. (2003). Incentives build robustness in BitTorrent. In Proceedings of the Workshop on Economics of Peer-to-Peer Systems.

(Crovella and Bestavros, 1996) Crovella, M. and Bestavros, A. (1996). Explaining world wide web traffic self-similarity. In Proceedings of the ACM SIGCOMM Conference.

(Fiscato et al., 2008) Fiscato, M., Costa, P., and Pierre, G. (2008). On the feasibility of decentralized grid scheduling. In Proceedings of the Workshop on Decentralized Self-Management for Grids, P2P, and User Communities, Venice, Italy.

(Foster and Kesselman, 1998) Foster, I. and Kesselman, C. (1998). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers.

(Jain et al., 1984) Jain, R., Chiu, D.M., and Hawe, W. (1984). A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems. DEC Research Report TR-301.

(Lal, 2007) Lal, N. (2007). Enforcing collaboration in a collaborative content distribution network. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands.

(Menasce and Almeida, 2001) Menasce, D. A. and Almeida, V. A. F. (2001). Capacity Planning for Web Services. Prentice Hall.

(Pierre and van Steen, 2006) Pierre, G. and van Steen, M. (2006). Globule: a collaborative content delivery network. IEEE Communications Magazine, 44(8):127-133.

(Popescu et al., 2003) Popescu, B. C., Crispo, B., and Tanenbaum, A. S. (2003). Secure data replication over untrusted hosts. In Proceedings of the 9th Workshop on Hot Topics in Operating Systems.

(Szymaniak et al., 2007) Szymaniak, M., Pierre, G., Simons-Nikolova, M., and van Steen, M. (2007). Enabling service adaptability with versatile anycast. Concurrency and Computation: Practice and Experience, 19(13):1837-1863.

(Weel, 2008) Weel, J. (2008). Gossiptron: Efficient sharing on the grid without central coordination. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands.

(Weisstein) Weisstein, E. W. Voronoi diagram. From MathWorld - A Wolfram Web Resource. --- http://mathworld.wolfram.com/VoronoiDiagram.html.