A Quantitative Comparison of Reputation Systems in the Grid

Jason D. Sonnek and Jon B. Weissman

Abstract— **Reputation systems have been a hot topic in the peer-to-peer community for several years. In a services-oriented distributed computing environment like the Grid, reputation systems can be utilized by clients to select between competing service providers. In this paper, we selected several existing reputation algorithms and adapted them to the problem of service selection in a Grid-like environment. We performed a quantitative comparison of both the accuracy and overhead associated with these techniques under common scenarios. The results indicate that using a reputation system to guide service selection can significantly improve client satisfaction with minimal overhead. In addition, we show that the most appropriate algorithm depends on the kinds of anticipated attacks. A new algorithm we've proposed appears to be the approach of choice if clients can misreport service ratings.**

Index Terms— **Grid Computing, Reputation Systems, Trust Models, Resource Scheduling**

I. Introduction

RESEARCHERS in the field of Grid computing envision a world in which individuals will be capable of dynamically pooling distributed resources for the purpose of solving complex problems. By utilizing excess computing resources spread across a number of organizations, demanding computational problems can be solved in a reasonable amount of time without the use of expensive dedicated supercomputers. In addition, Grid technologies enable the formation of virtual organizations capable of employing remote datasets, instrumentation, and other resources to address the challenges of business and science.

The Grid is already widespread in the scientific community. Physicists and biologists use the Grid as a distributed supercomputer, database, and as a means for collaboration. During the past few years, Grid-like systems have become increasingly popular outside of the scientific domain. For instance, systems like SETI@home, which utilizes idle cycles on donated personal computers, could be considered a type of Grid. In addition, companies like Sun and IBM currently use Grid technologies to provide computation time which can be rented by end users.

As the Grid continues to grow, it will become more common for individuals and companies alike to utilize the services of remote providers. These services may take the form of software programs, computational resources such as compute time and storage, or even access to specialized pools of information. By exercising the services offered by remote providers, end users save themselves the expense and inconvenience associated with the software, hardware, and resource management required to run these services locally.

In such an outsourced environment, service providers will compete to attract clients' business. This competition may encourage some service providers to exaggerate the capabilities of their services. Furthermore, clients run the risk of interacting with malicious providers, whose goals may range from disrupting the system to stealing the clients' time and money. For clients to feel comfortable participating in such an environment, mechanisms for minimizing the associated risks are required.

One well known technique for dealing with unreliable service providers is to submit redundant requests to multiple providers. Popular outsourced computing platforms such as SETI@home and BOINC use redundancy/voting schemes to deal with compute providers who misreport results. However, these schemes are very expensive in terms of the number of resources required to service a request. A more efficient approach would be to use available information about service providers to determine whether or not they are reliable.

To help solve this problem, researchers have introduced the concept of a reputation system. In a reputation system, statistics are maintained about the performance of each entity in the system, and these statistics are used to infer how entities are likely to behave in the future. Numerous reputation systems have been proposed for peer-to-peer networks [1, 2, 5, 6, 8, 9, 10, 11]. However, only a handful of systems have been proposed for the Grid [3, 4]. Furthermore, there has never been (to our knowledge) a study which quantitatively

Manuscript received June 3, 2005. This work was supported in part by National Science Foundation CNS-0305641 and ITR-0325949 and Department of Energy DE-FG02-03ER25554.

Jason D. Sonnek was a M.S. student in the Computer Science department at the University of Minnesota, Minneapolis, MN 55455, USA. He is now with Sandia National Laboratories, Albuquerque, NM 87185, USA. (phone: 651-209-7621; email: jsonnek@gmail.com).

Jon B. Weissman is an associate professor in the Computer Science department at the University of Minnesota, Minneapolis, MN 55455, USA. (email: jon@cs.umn.edu).

compares the accuracy and efficiency of these proposals against one another.

We have designed and implemented a Grid framework intended to model the interactions between clients and service providers. Within this framework, we have implemented a reputation system which can select from several different reputation algorithms to rank service providers. These reputation ratings are used by clients to choose service providers which are most likely to provide them with high-quality service (high likelihood of correct execution).

In this paper, we compare the effectiveness of these algorithms against a base system which lacks a reputation mechanism. We measured the benefit, from the client's perspective, of the recommendations provided by the reputation system for each of the algorithms. We also compare the relative cost of obtaining a recommendation using the different reputation algorithms, and determine how well each algorithm scales as the number of clients and service providers increases. The results indicate that if service providers are unreliable, using a reputation system to guide service selection can significantly improve client satisfaction, with minimal overhead. In addition, we show that the most appropriate algorithm depends on the kinds of anticipated attacks. In particular, a reputation system that uses global ratings works better if clients are honest, while a system that uses local ratings is better at thwarting misbehaving clients. A proposed new reputation algorithm appears to be the approach of choice if clients can misbehave.

II. MODEL

The services-oriented Grid model selected for this study is illustrated in Figure 1. We assume there are multiple service providers which offer competing services. Since these service providers may utilize resources which span organizational boundaries, we chose to apply the reputation system to individual services, as opposed to the organizations which provide them.

In contrast to a peer-to-peer system, the clients in our model are 'outside' the system. In other words, we assume that the clients do not also act as providers. In addition, these clients may not belong to any of the organizations which provide services in the Grid.

For simplicity, we assume a centralized, trusted reputation server which is visible to all clients and service providers. In a real system, there would likely be multiple reputation servers to improve reliability and availability. In addition, the question of where the reputation servers are located must be addressed. The reputation servers may be hosted within the Grid environment as a service to clients, though steps would need to be taken to ensure that service providers could not tamper with the ratings. Alternatively, the clients themselves could store the reputation ratings, distributing them amongst themselves using secure, anonymous storage [12]. We also assume that clients are able to verify the results they receive from service providers (this is how they will form opinions of the provider). In reality, this is a difficult problem for computational services, although some verification techniques have been proposed [7].

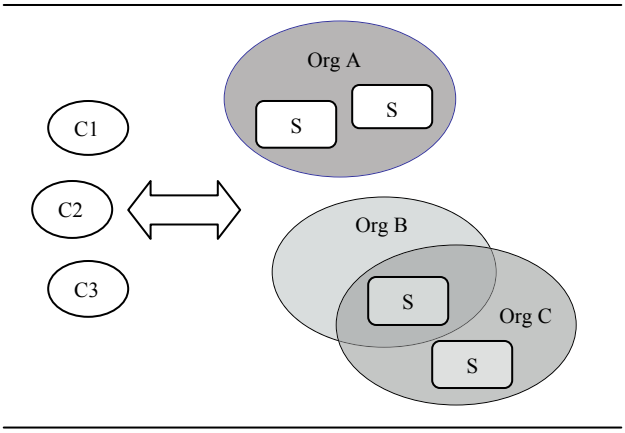

Figure 1: Grid Model. In this simple example, there are three different organizations participating in the Grid: A, B, and C. Within these three organizations, there are four competing services (S), one of which utilizes resources from multiple organizations. In such an environment, the clients (C1, C2, and C3), could benefit from some guidance regarding the reliability of the service providers.

III. REPUTATION ALGORITHMS

In this section, we will compare and contrast a number of published algorithms which we have adapted for use in the Grid model described above. We also present a new reputation algorithm which we've developed. The algorithms we selected were chosen due to their simplicity, low overhead, and appropriateness for the services-oriented Grid environment.

A. Simple Feedback

In spite of its simplicity, Ebay's feedback mechanism [14] is arguably the most successful reputation system to date. In this system, when a transaction is completed, each user is given the opportunity to rate the other party with regard to the quality of the transaction. A rating can be either +1 for satisfied, -1 for dissatisfied, or 0 for neutral; a user's feedback score is the sum of these individual ratings. Potential users of the system can use these scores as a heuristic for determining the risk involved in dealing with other users. The explosive growth of Ebay over the past decade is a testament to the effectiveness of this reputation system, since Ebay relies on its users' willingness to trust a perfect stranger. We implemented a simplified variant of this system, which we call the *Simple Feedback (SF)* algorithm, for comparison against the other techniques. In the SF algorithm the client returns feedback of either +1 (correct result received) or -1 (incorrect result

received) after each transaction. The rank of a provider is simply the sum of these ratings:

$$\text{Rank} = \sum ratings\ .$$

B. Beta Feedback

In [9], Jøsang and Ismail suggest a reputation system based on the beta density function. The beta density function is governed by two parameters, α and β. Given these parameters, we can use the resulting beta distribution to determine the probability of a binary event. If we let α = r + 1, where r is the number of positive outcomes observed thus far, and we let β = s + 1, where s is the number of negative outcomes, then the expected value of the beta distribution is approximately the number of positive outcomes observed over the total number of observations. This function is appealing as it is very similar to our intuitive understanding of reputation – the likelihood of receiving a satisfactory result from a provider is proportional to the fraction of satisfactory results received in the past. We implemented a slight variant of this technique, which we call the *Beta Feedback (BF)* algorithm. Upon receipt of a result from a service provider, the client verifies the result, and returns a feedback rating of +1 or -1 to the reputation server. The reputation server maintains aggregate *r* and *s* values for each service provider, and computes a rank using (1). In addition, the values are periodically discounted, so that more recent ratings have a greater impact on the provider's rank.

$$\text{Rank} = \frac{r+1}{r+s+2} \qquad (1)$$

C. Weighted Feedback

Azzedin and Maheswaran proposed a reputation system for Grid computing [4] which is based around a formal model of the way in which trust relationships are established in other contexts. In their model, the 'trust' which one entity has for another is a function of the direct experience between the two entities and their respective reputations. By differentiating between direct experience and reputation, the authors allow different entities to weight these two components as desired. For instance, suspicious entities may choose to give more weight to their personal experience with clients, under the belief that other's ratings cannot be trusted.

Our adaptation of this approach, which we call *Weighted Feedback (WF)*, differs from the Beta Feedback algorithm primarily in the separation of direct experience from second-hand information. The reputation server maintains separate *r* and *s* parameters for each client involved in the system. The rank of a service provider is computed by calculating the rating based on the requesting client's experience and the rating based on the feedback from all other clients, and then computing a linear combination of these two values (2). This approach also makes use of a decay function to give more weight to recent ratings.

$$\text{Rank} = \alpha F + \beta R \qquad (2)$$

$$F = \frac{r}{r+s} \quad \text{(r,s : requesting client)}$$

$$R = \frac{\sum r}{\sum r + \sum s} \quad \text{(r,s : all other clients)}$$

The next three algorithms are designed specifically to deal with the problem of dishonest ratings in reputation systems. These systems are intended to minimize the impact of *badmouthers* (clients who give unfair negative ratings to service providers) and *ballot stuffers* (clients who give unfair positive ratings to service providers).

D. Beta Filtering Feedback

In [13], the authors consider statistical techniques for filtering out dishonest raters in the beta algorithm discussed in (b). The general idea behind this approach is to identify clients whose ratings for a given service provider fall outside the normal range of values observed for that client. This can be achieved using statistical filtering as follows:

```
For each provider
   a. Compute provider rank using BF
   b. Add all clients to client-list
   c. For each client in client-list
        I. Compute beta distribution of
           client's ratings of provider
       II. If provider rank from (a) is less
           than the 1% quantile or greater
           than the 99% quantile of beta
           distribution from (I), remove
           client from client-list
   d. Repeat step (c) until client-list no
      longer changes
   e. Compute provider rank using only clients
      in client-list
```

We added this technique to the BF algorithm discussed earlier, and have named the modified algorithm *Beta Filtering Feedback (BFF)*.

E. Beta Deviation Feedback

An alternative approach to filtering out dishonest raters is to use deviation testing, as suggested in [5]. In this system, a beta distribution is used to represent the trustworthiness of a provider from *each* client's perspective. That is, rather than maintaining a single set of beta parameters for all clients, the reputation server maintains a set of beta parameters for each client. The parameters are updated as follows:

1. Whenever a client has a firsthand experience with a provider, it updates its parameters, and broadcasts this rating to all other clients.

2. When a client receives a rating from another client, it updates its parameters if it believes the client that sent the rating is trustworthy.

The trustworthiness of a client is determined using statistical deviation testing. Specifically, each client maintains a second set of beta parameters which represent the likelihood of another client giving an honest rating. Suppose client c_1 receives a rating from client c_2. Then, c_1 will use the following algorithm to determine whether or not to incorporate c_2's rating of p_1 in its score of p_1:

```
If honesty(c2) > t, include rating
Else, compute deviation test
   a. If  rankc1(p1)-rankc2(p1)  <  d,  include
      rating, and increment trust score
   b. Otherwise, exclude rating, and decrement
      trust score
```

The rank is computed using the *r* and *s* scores of the requesting client as in III.B. The parameters *d* and *t* can be adjusted to reflect the degree to which clients wish to expose themselves to risk in their dealings with service providers. Throughout the rest of the paper, we refer to this technique as the *Beta Deviation Feedback (BDF)* algorithm.

F. Selective Weighted Feedback

Finally, we considered updating the *Weighted Feedback* algorithm with a filtering technique of our own. The goal of this algorithm was to maintain the ability to filter out dishonest raters while introducing as little additional complexity as possible. Towards this end, we modified the *Weighted Feedback* algorithm so that it only uses only the ratings from a select number of the most trusted clients. Each client maintains a vector of honesty ratings of the other clients. When computing the rank of a provider, it asks the 5 most-honest for their ratings, and factors them into the rank using the *Weighted Feedback* algorithm. To ensure that the client doesn't get stuck with a sub-optimal trusted set, it occasionally selects a client at random rather than selecting one of the top clients. The honesty ratings are updated as follows:

```
For each client in recommender list
  If result was positive
    If client's rating  was >= 0.5
     increment trust rating
    Else
      decrement trust rating
  If result was negative
    If client's rating was < 0.5
      increment trust rating
    Else
      decrement trust rating
```

In other words, if a client gave a high rating to a provider and we received a good result, our trust for that client increases. Otherwise, it decreases. For a negative result, we perform the inverse operation. Since it uses rating information from a selected subset of clients, we call this algorithm *Selective Weighted Feedback (SWF).*

Some of the key differences between the six algorithms we've discussed are summarized in Table 1. As we mentioned earlier, BFF, BDF, and SWF use different techniques to determine which ratings to ignore when computing a provider's rank, under the assumption that some clients will submit dishonest ratings. In contrast, SF, BF, and WF use all of the available rating information. Another key distinction is that SF, BF, and BFF attempt to calculate a global rating for a provider. In other words, the rating is the same no matter which client requested it. In systems which use the WF, BDF, or SWF algorithms, different clients may have different ratings of the same provider, since these algorithms rely heavily on the client's first-hand experience with a provider.

Name	Rating type	Filtering type
Simple Feedback	Global	None
Beta Feedback	Global	None
Weighted Feedback	Local	None
Beta Filtering Feedback	Global	Ignore clients whose ratings are far from average global rating
Beta Deviation Feedback	Local	Ignore ratings which deviate from local ratings
Selective Weighted Feedback	Local	Ignore ratings from all but a few most 'trusted' clients

Table 1: Key differences between algorithms

IV. IMPLEMENTATION

To compare the performance of the rating algorithms, we designed and implemented the services-oriented Grid model illustrated in Figure 2.

The reputation server listens for registration messages from service providers and rating requests from clients. Upon receipt of a registration message, the reputation server allocates data structures which are used to store reputation data about that provider. When it receives a recommendation request, the reputation server computes a list of the providers offering the requested service. Next, it computes a rank for each provider in the list based on the reputation algorithm which is being used. Finally, it sorts the list by rank, and returns it to the client. Since we are only interested in measuring the accuracy and performance of the reputation

system we make a few simplifying assumptions. First, we assume that the reputation server is always honest, and that it is impervious to subversion. In a real system, one would have to employ techniques to ensure the integrity and authenticity of the reputation server. In addition, we assume there is only a single reputation server. To improve reliability and availability, one would likely use multiple reputation servers in a live implementation. Finally, we assume that the reputation server can uniquely identify clients and service providers, so that attacks based on impersonating other entities (e.g., Sybil attacks) are not possible. Again, for this assumption to hold in the real world, one would need to rely on cryptographic authentication techniques. Since we are only interested in reputation costs and benefits, the services we have deployed do not carry out any real service, though they run live and participate in each reputation protocol.

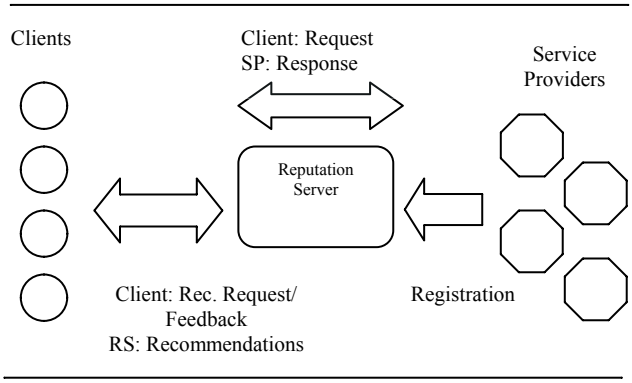

Figure 2: Grid Implementation

Upon receipt of a client request, a service provider returns either a correct or incorrect result, depending on the provider's strategy. In this paper, we consider two different service provider strategies. Providers with a 'fixed' strategy have some internal probability (which is unknown to the client) which governs whether or not they return correct results. For instance, some of the providers return a correct result 95% of the time, and some of the providers return a correct result 20% of the time. The intent of this approach was to model the uncertainty involved in dealing with remote service providers in a distributed environment. For all of the experiments presented in this paper, the number of 'fixed' service providers is evenly distributed between reliable and unreliable providers.

Providers with a 'variable' strategy also have an internal probability, but this probability changes over time due to some external factors. When comparing the benefit of different reputation systems, we introduced some percentage of *malicious* providers into the system. Malicious providers return the correct result to all clients until they have built up a strong positive reputation. Then, they start returning incorrect results all of the time. These services are intended to represent providers who may try to cheat the system by building up a good reputation and then starting to misbehave.

To measure the relative benefit of each of the different algorithms, we chose to measure their impact on the number of requests a client must make before receiving a correct result. To accomplish this, we designed the client's resource selection policy to be guided by recommendations provided by the reputation server. The idea of reputation-based resource selection in the Grid was proposed in [3].

When a client is started, it repeatedly generates requests. First, the client queries the reputation server for a list of nodes which provide the desired service. A recommendation list is returned to the client, sorted by reputation score. Then, the client selects a provider from the list, and issues a request to that provider. Based on the value that is returned by the provider, the client submits a feedback rating to the reputation server. If a client receives an incorrect result from a service provider, it selects another provider from the recommendation list, and repeats the above process. This allows us to measure the effectiveness of the reputation system in terms of the average number of requests that a client has to make before receiving a correct result.

To ensure that new providers are given a chance to prove that they are reliable, the client selects a top-rated service provider from the recommendation list with high probability, and selects a provider at random with some small probability. This also ensures that providers which may have a bad reputation due to unfair ratings have an opportunity to redeem themselves. For the experimental results presented below, a random provider is chosen 10% of the time.

We model three different types of clients: honest, badmouthing, and ballot-stuffing. Honest clients always tell the truth about whether or not they received a correct result. Badmouthing clients always claim they received an incorrect result from some targeted provider, and ballot-stuffing clients always say that they received a correct result from some targeted provider. In all of the experiments, the clients are honest unless otherwise specified.

In our experiments, we ran the reputation server on one machine, and all the other entities on a separate machine. Since we are measuring the performance of the reputation server, this ensures that it is isolated from the clients and service providers (as it likely would be in a real system). Since the accuracy results depend only on the algorithms, and not on the location of system entities, it is not necessary for each client and service provider to have a dedicated machine. As stated earlier, a few of the algorithms depend on certain parameter settings. For the algorithms we adapted, we used the default values suggested by the original authors. If no default was listed, the parameters were determined experimentally. The settings of algorithm parameters for all experiments are listed in Table 2.

Algorithm	Parameters
BDF	w = 0.1, d = 0.2, t = .25
WF	$\alpha = 0.9, \beta = 0.1$
SWF	$\alpha = 0.9, \beta = 0.1$

Table 2: Default Parameter Settings

V. RESULTS

In this section, we will quantitatively compare the algorithms presented in section III in terms of the benefit and overhead they introduce into the system. For our first experiment, we fixed the number of clients at 50 and the number of service providers at 40. Each client makes 1000 requests to the service provider, so the average results presented for each algorithm are for 50,000+ requests. We varied the percentage of malicious service providers from 0 to 70%, and measured the effectiveness of the reputation-guided resource selection under each of these scenarios. Each of these experiments was run five times, to filter out some of the variance in the results that is caused by the uncertainty inherent in the simulated workload.

Figure 3 compares each of the six algorithms to a simple random selection policy. It is clear that using any of the reputation systems to guide resource selection is far superior to a random selection policy. Additionally, as the percentage of unreliable or misbehaving service providers increases, the disparity becomes even larger. This establishes the benefits of a reputation system for request-reply service selection even under modest levels of malicious service behavior.

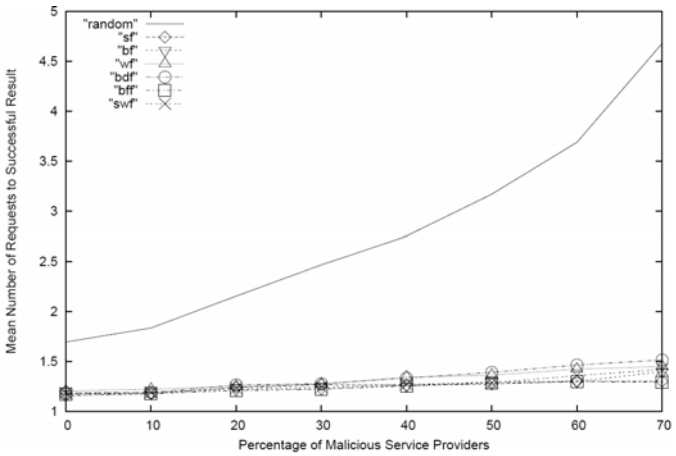

Figure 3: Algorithm Comparison

In Figure 4, we've eliminated the random selection policy from the plot, to make the difference between the reputation functions clearer. The *Simple Feedback* and *Beta Filtering Feedback* algorithms are about 10% better, on average, than the two worst algorithms, *Weighted Feedback* and *Beta Deviation Feedback*. The reason for this difference is that the SF, BF, and BFF algorithms assign global ratings to each of the providers. Thus, every client benefits from all of the other client's experiences. Since all of the clients are honest in this environment, the reputation functions which make use of the most information are more accurate.

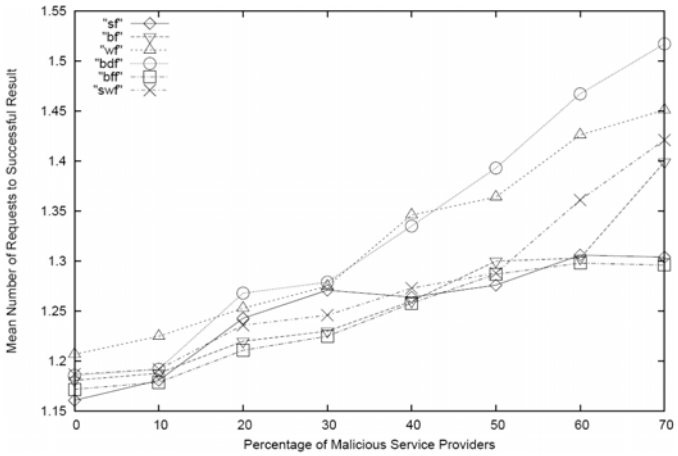

Figure 4: Algorithm Comparison Close-up

In contrast, the algorithms which attempt to screen out dishonest ratings don't perform as well. This is due to the fact that when a client makes a request, it doesn't have as much information about the providers' past history. The BDF algorithm in particular suffers from some amount of false positives in its filtering algorithm, which causes some honest raters to be filtered out. If one could ensure that all of the ratings that were returned by clients were honest, it seems that simple reputation algorithms like SF and BF are sufficient.

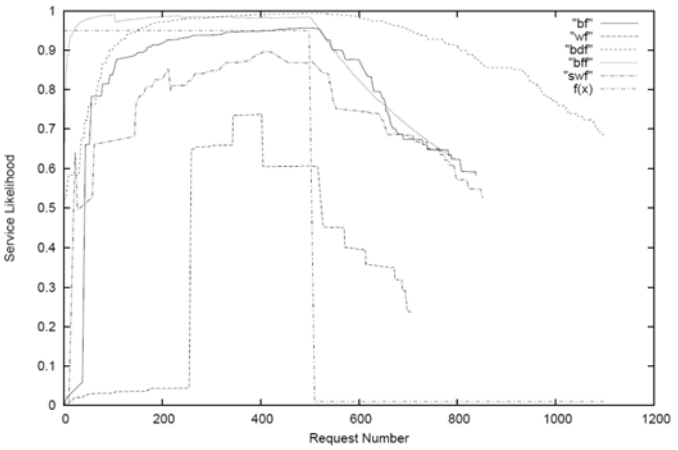

Figure 5: Accuracy Comparison

In Figure 5, we've plotted the reputation scores returned by each of the different algorithms for a given provider as the number of requests to that provider increases. The function f(x) in the graph is the actual internal probability that this provider uses when deciding whether or not to return a correct result. In this example, the provider gives a correct result with high probability until its 500th request, at which time it starts becoming very unreliable. Although each of the algorithms is different, they all learn the change in probability and the reputation drops off with approximately the same slope. In fact, the one which drops off the slowest (BDF) was also the

worst performer in this experiment. This suggests a correlation between the learning rate and the utility of the algorithm.

In the next two experiments, we consider what happens if dishonest clients are introduced into the system. For these experiments, the percentage of malicious service providers was fixed at 20%, and the number of clients and service providers remained the same.

First, we considered the effect of 'badmouthing' clients. As stated earlier, badmouthing clients always return negative feedback about the service providers they are targeting. For this experiment, we vary the percentage of badmouthing clients from 0 to 50%, and measure the impact that they have on the performance of the reputation functions. The badmouthing clients behave in the same manner as the honest clients (submit 1000 requests), but they always return negative feedback about a selected group of very reliable providers. They are intended to represent malicious clients whose goal is either to disrupt the system, or perhaps to profit by targeting their competitors. Note that in this experiment, we compared the three algorithms which attempt to filter out dishonest raters to the *Simple Feedback* algorithm, which was one of the best algorithms from the previous experiment.

Figure 6 shows that even with only a small number of badmouthing clients in the system, the *Simple Feedback* algorithm is no longer effective. The BDF algorithm does a good job of minimizing the dishonest client's impact, and our SWF algorithm looks especially good for filtering out dishonest raters. This test uncovered some interesting properties of the BFF algorithm which are worth mentioning. As stated earlier, the values in Figure 6 are an average across multiple runs. In the individual trials with BFF, it performed about as well as SWF in half of the trials, and performed about the same as SF in the other half!

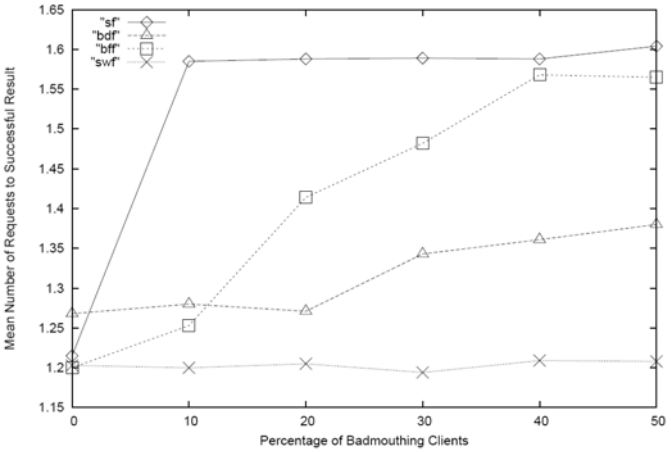

Figure 6: Algorithm Comparison - Badmouthing Clients

The reason for this disparity lies in the global filtering function utilized by BFF. If a reliable provider gets a bad reputation due to the badmouthing clients, it is never able to recover. Since it will rarely be selected as a provider in the future, the few honest ratings which it receives are filtered out as being unusual. BDF doesn't suffer from this problem, because its filtering is based on the client's local observations, rather than the global view of all clients.

Next, we investigated the impact that 'ballot-stuffing' clients have on the system. These clients always return false positive ratings about selected service providers. In Figure 7, we've plotted the results for the same four algorithms used in the previous experiment. In this case, the BDF, BFF, and SWF algorithms all have very similar results. As the percentage of ballot-stuffing clients becomes extremely high, the performance of BFF starts to degrade, due to the fact that some honest clients are misled by the false positive ratings given to the ballot-stuffer's targets. However, the negative ratings returned by the honest clients ensure that the malicious providers can never get a very good reputation, in spite of the ballot-stuffers' efforts.

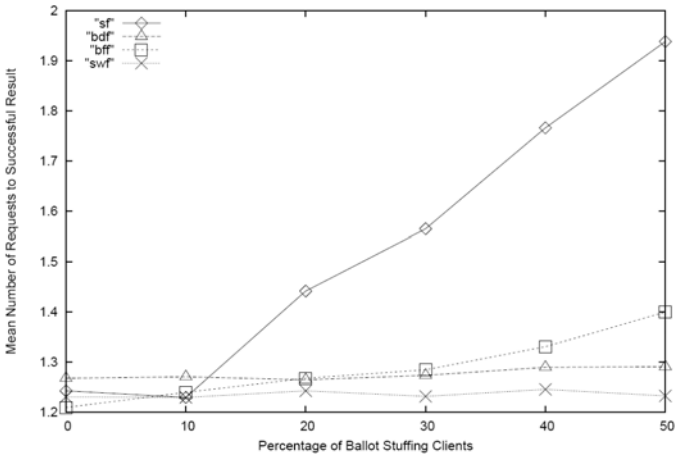

Figure 7: Algorithm Comparison – Ballot-Stuffing Clients

These results strongly suggest that for an environment with a significant percentage of dishonest raters, it is best to select a reputation algorithm which utilizes local ratings (e.g., BDF or SWF) as opposed to one which uses global ratings. Considering the results as a whole, it is clear that there is no 'best' algorithm for all scenarios. In systems where the ratings supplied by clients cannot be trusted, SWF shines due to its heavy reliance on the client's first-hand experience. However, this algorithm does not perform well when all raters are honest, because it uses a limited amount of information when making decisions. To obtain the most benefit, algorithm selection ought to be guided by some understanding of the environment into which the reputation system is to be deployed.

Finally, we measured the relative performance difference between the different algorithms. Clearly, the algorithms which employ filtering techniques will impose more overhead on the reputation server. The goal of the following

experiments was to measure the amount of overhead introduced, and to determine if it will impact the client's experience.

We measured the performance of each algorithm by comparing the amount of time required to generate a recommendation, and the amount of time required to update ratings after feedback was received, as the number of clients and service providers was scaled up from 10's to 100's.

For a client pool increasing from 50 to 500, and a service provider pool increasing from 25 to 100, the reputation server was able to obtain a recommendation in 10 ms or less. In a live system, this processing time will likely be minor compared to the network overheads involved. Thus, we conclude that for a system of this scale, implementing a reputation system will have a minimal impact on the client's experience. However, for some of the algorithms, the reputation server did experience considerably more computational load during the update phase. Table 3 shows the results for each of the algorithms as we scale the number of clients up from 50 to 500, with the number of service providers fixed at 25.

Most of the algorithms didn't require any significant processing until the number of clients increased to 500. At this point, the BF, WF, and SWF average run times were in the 10's of milliseconds. Due to large variance in the measurements, we conclude that most of this overhead is due to wait time associated with shared data structures.

	Mean Run Time (ms)		
Algorithm	50 clients	100 clients	500 clients
SF	0.03	0.04	0.04
BF	0.27	4.25	51.19
WF	0.73	1.06	51.37
BDF	0.10	0.20	0.70
BFF	27.71	86.57	235.99
SWF	1.29	2.92	49.06

Table 3: Performance Comparison

SF and BDF both scaled extremely well (less than 1 ms for all client sizes) due to their lack of shared data structures. BFF, however, has considerably higher overhead for all of the client sizes we measured, and variance measures suggest that these run times are fairly uniform. While its results were impressive in some of the earlier experiments, it relies on a complex filtering function to obtain those results. Thus, we conclude that although its accuracy is very high, BFF does not scale well due to its inherent complexity.

VI. Conclusion

In this paper, we have investigated the effectiveness of reputation systems in a services-oriented Grid where clients request services from competing service providers. We have implemented adaptations of several algorithms from the literature and an algorithm of our own design. Using this Grid model, we have performed a quantitative comparison of both the accuracy and overhead associated with these techniques under common scenarios. Our results show that using a reputation system to guide service selection can significantly improve client satisfaction with minimal overhead. Furthermore, our results indicate that depending on the properties of the system, there is a significant difference in the performance of different ranking algorithms. In particular, a reputation system that uses global ratings works better if clients are honest, while a system that uses local ratings is better at thwarting misbehaving clients. Our evaluation can help system architects to select an algorithm which meets the needs of their environment.

References

[1] K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System", in *Proceedings of the Tenth International Conference on Information and Knowledge Management*, 2001

[2] Z. Abrams, B. McGrew, and S. Plotkin, "Keeping Peers Honest in EigenTrust", in *Second Workshop on the Economics of Peer-to-Peer Systems*, Harvard Univ., June 4-5 2004

[3] B. K. Alunkal, I. Veljkovic, G. von Laszewski, and K. Amin, "Reputation-Based Grid Resource Selection", in *Workshop on Adaptive Grid Middleware (AGridM 2003)*, New Orleans, LA, USA, Sept. 28, 2003

[4] F. Azzedin and M. Maheswaran, "Evolving and Managing Trust in Grid Computing Systems", in *Proceedings of IEEE Canadian Conference on Electrical & Computer Engineering*, pp. 1424-1429, May 2002

[5] S. Buchegger and J-Y. L. Boudec, "A Robust Reputation System for P2P and Mobile Ad-hoc Networks", in *Second Workshop on the Economics of Peer-to-Peer Systems*, Harvard Univ., June 4-5 2004

[6] E. Damiani, S. De Capitani di Vimercati, S. Parabosci, P. Samarati, and F. Violante, "A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks", in *Proceedings of the Ninth ACM Conf. on Computer and Communications Security*, pp. 207-216, 2002

[7] W. Du, M. Mangal, and M. Murugesan, "Uncheatable Grid Computing", in *Proceedings of the 24th International Conference of Distributed Computing Systems*, pp. 4-11, 2004

[8] D. Dutta, A. Goel. R. Govindan, and H. Zhang, "The Design of A Distributed Rating Scheme for Peer-to-peer Systems", in *Workshop on the Economics of Peer-to-Peer Systems*, U.C. Berkeley, June 2003

[9] A. Jøsang and R. Ismail, "The Beta Reputation System", in *Proc. of the 15th Bled Conf. on Electronic Commerce*, Bled, Slovenia, June 2002

[10] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks", in *Proc. of the 12th International Conf. on World Wide Web*, pp. 640-651, 2003

[11] L. Mui, M. Mohtashemi, C. Ang. P. Szolovits, and A. Halberstadt, "Ratings in Distributed Systems: A Bayesian Approach", in *Workshop on Information Technologies and Systems*, New Orleans, LA, USA, December 15-16, 2001

[12] A. Singh and L. Liu, "TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems", in *Proceedings of the IEEE International Conference on P2P Computing,* Linköping, Sweden, September 2003

[13] A. Whitby, A. Jøsang, and J. Indulska, "Filtering Out Unfair Ratings in Bayesian Reputation Systems", in *The Icfain Journal of Management Research*, Vol. 4, No. 2, pp. 48-64, February 2005

[14] Ebay, http://www.ebay.com