

Jiminy: A scalable incentive-based architecture for improving rating quality

Evangelos Kotsovinos¹, Petros Zerfos¹, Nischal M. Piratla¹,
Niall Cameron², and Sachin Agarwal¹

¹ Deutsche Telekom Laboratories,
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
`{firstname.lastname}@telekom.de`

² Pembroke College,
Cambridge CB2 1RF, UK
`niall.cameron@cantab.net`

Abstract. In this paper we present the design, implementation, and evaluation of Jiminy: a framework for explicitly rewarding users who participate in reputation management systems by submitting ratings. To defend against participants who submit random or malicious ratings in order to accumulate rewards, Jiminy facilitates a probabilistic mechanism to detect dishonesty and halt rewards accordingly.

Jiminy’s reward model and honesty detection algorithm are presented and its cluster-based implementation is described. The proposed framework is evaluated using a large sample of real-world user ratings in order to demonstrate its effectiveness. Jiminy’s performance and scalability are analysed through experimental evaluation. The system is shown to scale linearly with the on-demand addition of slave machines to the Jiminy cluster, allowing it to successfully process large problem spaces.

1 Introduction

Reputation management systems (RMSs) allow participants to report their experiences with respect to past interactions with other participants. RMSs are often provided by retailer web sites, on-line movie review databases, auction systems, and trading communities. However, as identified in [12], information within such systems may not always be reliable. Many participants opt not to submit ratings, as there is little incentive for them to spend time performing the rating task — especially if interactions are frequent and participants expect utility standards of service. Furthermore, participants tend to report mostly exceptionally good or exceptionally bad experiences as a form of reward or revenge respectively. Additionally, ratings are often reciprocal, as underlined by the observation that a seller tends to rate a buyer after the buyer rates the seller [17].

In our previous work we proposed Pinocchio [12], an incentive model where participants are explicitly *rewarded* for submitting ratings, and are *debited* when they query the RMS. Providing explicit incentives a) increases the quantity of

ratings submitted and b) reduces the bias of ratings by removing implicit or hidden rewards, such as revenge or reciprocal ratings. To prevent participants from submitting arbitrary or dishonest feedback with the purpose of accumulating rewards, Pinocchio features the credible threat of halting rewards for participants who are deemed dishonest by its probabilistic *honesty estimator*.

In this paper we present *Jiminy*, a distributed architecture that employs the Pinocchio model to build a scalable system for providing participation incentives. This work makes the following contributions:

- It presents the design, implementation, and cluster deployment of Jiminy, including an algorithmic implementation of the Pinocchio mathematical model, suitable for a clustered architecture.
- It analyses a large data set (from the GroupLens³ project) of one million movie ratings submitted by human users. It does so in order to verify the validity of the assumptions based on which our model is designed, regarding the distribution of our honesty estimator and the independence of ratings.
- It demonstrates the effectiveness of the Jiminy honesty assessment algorithm by showing that it can detect four different types of dishonest users injected into the GroupLens data set.
- It demonstrates experimentally that the Jiminy cluster can scale to process large problem spaces — i.e., large numbers of participants and ratings — at run-time, as the cluster’s performance increases almost linearly with the addition of more slave machines.

The rest of the paper is organised as follows. A brief outline of the Pinocchio reward and honesty assessment scheme is provided in Section 2. The design, implementation, and deployment of Jiminy in a cluster — including an algorithmic realisation of the Pinocchio model and extensions for its efficient implementation in a distributed environment — is described in Section 3. The effectiveness of the Jiminy algorithms and the scalability of the clustered system is evaluated in Section 4. Related work in trust management and incentive schemes is presented in Section 5. Finally, our conclusions and future work are discussed in Section 6.

2 Background

The Pinocchio model provides a mechanism for the assessment of the quality of ratings that are submitted by users⁴ of a reputation management system (RMS). It defines an *honesty metric* that quantifies in a probabilistic sense how honest a participant is in her ratings, and outlines a *reward model* that provides incentives for participants to submit honest ratings. In this section we provide brief background information on the Pinocchio model — a detailed description is provided in [12].

³ <http://www.grouplens.org>

⁴ we use the terms *user* and *participant* as equivalent throughout this paper.

2.1 Honesty metric

The honesty metric in the Pinocchio model is used to discourage participants from submitting random ratings in an attempt to accumulate rewards. Moreover, it detects participants who always submit average ratings, which convey little useful information to other users of the reputation management system. The entities that are reviewed by participants are referred to as *subjects*. When participants interact with a subject they form opinions about it and report these opinions to the RMS through a numeric score, or *rating*. The opined-about subjects comprise the set R . Sufficient number of ratings by different participants on a specific subject can be used to fit a probability distribution that corresponds to the ratings of all participants for that subject. This probability distribution can then be used to check the credibility of ratings submitted by participants for that subject.

Suppose participant u reported a rating value Q_s for subject s . We compute the log-probability of likelihood L_s of Q_s based upon the probability distribution of all ratings available for subject s . The more unlikely a rating value on a subject, the more negative L_s becomes:

$$L_s = \ln(\Pr(Q_s)) \quad (1)$$

We define a subset B , $B \subseteq R$, which contains all subjects rated by a given participant u . Summing over all subjects (elements of B) about which u has reported reviews, we obtain T_u :

$$T_u = \sum_{s \in B} \ln(\Pr(Q_s)) \quad (2)$$

These values alone are not sufficient for estimating honesty; they would be biased towards users who are continually submitting ratings with a high probability, i.e., the average opinion of the community. It is necessary to protect against this, thus the honesty estimator measures the deviation Z of random variable T_u from its mean value \bar{T} (considering all participants), and standard deviation $\hat{\sigma}$:

$$Z = \frac{(T_u - \bar{T})}{\hat{\sigma}} \quad (3)$$

$|Z|$ is used as the estimator, and is referred to as the *nose-length* of the participant in the Pinocchio model.

The calculation of the nose-length $|Z|$ of a participant from Equation 3 requires the scaling of her T_u value according to the total number of rating submissions the participant has made. Without the scaling process, all users are expected to provide same number of ratings, placing a limitation on the applicability of the model to real-world data sets. In Jiminy this aspect is taken into account in the definition of honesty metric, as discussed in the following section.

2.2 Reward model

Participants are encouraged to submit their experiences on subjects by being provided with a reward for each experience that they report. A credit balance

is kept for each participant, which is credited with a reward for each rating that she contributes, and debited for each query that she makes.

Rewarding a participant for the ratings she makes is also subject to whether she is deemed honest or not. When her nose-length rises above a certain threshold (*dishonesty threshold*), she is deemed dishonest and rewards for further ratings that she submits to the RMS are halted. For her rewards to be resumed, her honesty metric has to fall below the *honesty threshold*, and remain so for a specific period of time (*probationary period*). Pinocchio seeks to ensure that there is a substantial penalty for recurring dishonest behaviour, and, at the same time avoid being too strict to first-time cheaters. To achieve this, an exponential back-off scheme is followed for the value of the probationary period, which is doubled each time a participant is considered as being dishonest.

3 Design and Implementation

In this section we present the design and implementation of Jiminy. We describe the main components and operations of the system, and discuss the approach that we followed to realise its deployment in a distributed environment. The latter is necessary in order for Jiminy to scale to a large number of participants, each of which requires computationally intensive operations and increases the system load considerably.

Jiminy is a multiprocess, multithreaded application written in Java, which interacts with a MySQL backend database for storage and retrieval of system/user data. Figure 1 shows the main components of the system, along with their interaction with the RMS. The architecture of Jiminy follows the master-slave model for server design. The goal is to distribute the processing load that is involved in the calculation of the nose-lengths $|Z|$ of participants to multiple machines, and parallelise the operation so that it scales with the number of participants and ratings. The more slave machines are added to the system, the higher the possibility of bulk processing of user honesty updates.

Without limiting the applicability of Jiminy, we assume the RMS to be distributed, running a node on each Jiminy slave. This RMS architecture is similar to that of BambooTrust [13].

Master. Upon initialisation of the system, the main process on the master node starts a new thread that listens for incoming registration requests from slave nodes — operation 1 in Figure 1. Once several slaves have registered their presence with the master node, the latter assigns to each one of them a distinct subset of all users that participate in the ratings system, which is used to populate each slave’s local database — operation 2. The user subsets that are assigned to the slave machines are disjoint to eliminate contention for any given user profile on the master, minimise the load from queries on participant information submitted by slave machines to the master node, and also reduce network traffic.

Additionally, when the master node receives a query from the RMS regarding the trustworthiness of a user — operation 3, it acts as a dispatcher and forwards

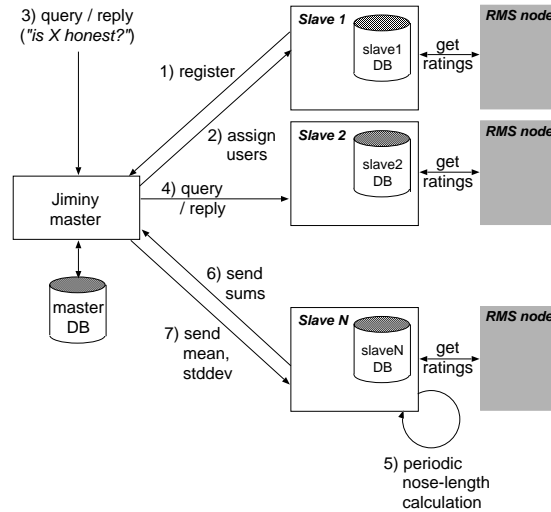


Fig. 1. An overview of the Jiminy clustered architecture

the request to the appropriate slave machine for retrieving the respective value — operation 4. Queries are encoded in XML format to allow interoperability with a variety of reputation management systems. Dispatching of queries is also handled by a separate thread to allow the main process to maintain an acceptable level of responsiveness of the system to user input. Lastly, the master also provides a graphical user interface, through which users of the system can perform queries on the honesty of participants, and set the system parameters such as honesty and dishonesty thresholds.

Slave. The main process (*Slave i , $i = 1..N$* in Figure 1) that runs on a slave node initially registers itself with the master — operation 1, and receives the subset of participants this slave will be responsible for, as well as system-wide variables — operation 2. It then listens for incoming query requests from the master. Queries are of several types such as requests for the credit balance of a user, notifications of a new rating to the RMS, requests for a trust value, etc. They are parsed and processed by a pool of threads that is started by the main slave process — operation 4.

Slave nodes also update the honesty metrics for their assigned participants, and calculate a user's position with respect to the reward model. This is performed by a separate thread that runs periodically on the slave, connects to the RMS node to receive aggregate information on the ratings for use in honesty calculations, and updates the honesty values of all participants who have in the past evaluated subjects that received a new rating — operation 5.

The Jiminy system makes use of persistent storage for storing intermediate results and general statistics on the participants and the subjects that are rated.

The aim is to avoid costly re-calculations upon system crash, and perform only incremental updates on the T_u values as new ratings are submitted to the RMS. System information such as honesty threshold, length of probationary period, mean and standard deviation of the T_u values, as well as histogram statistics for the rated subjects are stored in the local MySQL database on each slave machine.

3.1 Implementation details

At the heart of Jiminy lies the algorithm that periodically updates the T_u values and the nose-lengths $|Z|$ of the participants. The probability distribution $Pr(Q_s)$ of the ratings available for a given subject s (Equation 2) is estimated using the following formula for every rating ρ :

$$Pr(Q_s) = \frac{\# \text{ participants who assigned rating } \rho \text{ to } s}{\# \text{ participants who rated } s}$$

As stated in previous section, the calculation of the nose-length $|Z|$ of a participant from Equation 3 requires the scaling of her T_u value. Without this adjustment, a participant's T_u value is proportional to the number of her submissions, and in case it is different from the average number of ratings submitted per user she is deemed more dishonest. This is also intuitive in the sense that a participant with many ratings is more likely to have made dishonest ratings, however Jiminy is interested in the rate of disagreement, not the total number of its occurrences. To account for this fact, Equation 3 is modified as follows:

$$Z = \frac{\frac{T_u}{\# \text{ ratings made by } u} - \bar{T}}{\hat{\sigma}} \quad (4)$$

The pseudocode for the calculation of T_u values is shown in Algorithm 1.

As new ratings get submitted to the reputation management system, users who have reviewed subjects that are rated by the new ratings need to have their T_u values updated (variable *AffectedUsers* of Algorithm 1). For each one of these users, the algorithm finds the ratings that affect her T_u value and accordingly adjusts it based on whether the user rated the subject with the same rating as the one carried in the new rating (lines 11–15 of Algorithm 1). Since T_u values do not change dramatically from one rating to another, the algorithm runs periodically to reduce processing overhead, and waits for several new ratings to accumulate in the RMS. New ratings are determined using a timestamp that is associated with each rating that is received.

Each slave machine updates the nose-lengths of the users that have been assigned to it. To better distribute the user space to the group of slaves, the master assigns new and existing users to slaves according to the formula: $slave_number = (userid \% n)$, when n is the number of slaves present. Failure of slave machines is handled by the master using the Java exception handling facility, however a more robust fail-over mechanism still remains part of our future work.

Algorithm 1 Update T_u Values()

Require: New ratings added to RMS since last update

```
1:  $AffectedUsers \leftarrow$  Find users that have rated subjects that appear in new ratings
2: for (each user  $u$  in  $AffectedUsers$ ) do
3:    $UserSubjects \leftarrow$  Find subjects rated by user  $u$ 
4:    $NewUserRatings \leftarrow$  Find new ratings about  $UserSubjects$ 
5:   for (each new user rating  $i$  in  $NewUserRatings$ ) do
6:      $Subject \leftarrow$  Subject rated by  $i$ 
7:      $UserRating \leftarrow$  Rating about  $Subject$  by user  $u$ 
8:      $NumberSame \leftarrow$  Number of ratings about  $Subject$  equal to  $UserRating$ 
9:      $TotalSubjectRatings \leftarrow$  Number of ratings that rate  $Subject$ 
10:     $T_u \leftarrow T_u - (\log(NumberSame) - \log(TotalSubjectRatings))$ 
11:    if ( $UserRating =$  rating of rating  $i$ ) then
12:       $T_u \leftarrow T_u + (\log(NumberSame + 1) - \log(TotalSubjectRatings + 1))$ 
13:    else
14:       $T_u \leftarrow T_u + (\log(NumberSame) - \log(TotalSubjectRatings + 1))$ 
15:    end if
16:  end for
17: end for
```

Last, the calculation of nose-lengths $|Z|$ requires the mean \bar{T} and standard deviation $\hat{\sigma}$ of the T_u values of all the N participants. From the formulas for the mean and standard deviation we have:

$$\bar{T} = \frac{1}{N} \sum_{u=1}^N T_u \quad (5)$$

and:

$$\hat{\sigma} = \sqrt{\frac{1}{N-1} \sum_{u=1}^N (T_u - \bar{T})^2} \quad (6)$$

By substituting (5) into (6) and after simple algebraic manipulations, we get:

$$\hat{\sigma} = \sqrt{\frac{1}{N(N-1)} \left(N \sum_{u=1}^N T_u^2 - \left(\sum_{u=1}^N T_u \right)^2 \right)} \quad (7)$$

Each slave transmits the sum and the sum-of-squares of T_u values for its participant set to the master — operation 6 in Figure 1. The master then calculates the mean and standard deviation for all the participants, and disseminates the results back to the slaves for further use in estimating $|Z|$ — operation 7.

4 Evaluation

In this section we present a three-fold evaluation of our architecture. We first analyse the GroupLens data-set to ensure that the assumptions made by our

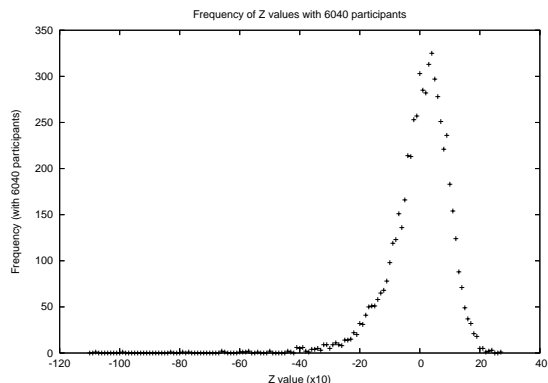


Fig. 2. Distribution of Z values (x10)

model about the distribution of nose-lengths and independence of ratings hold. Furthermore, we demonstrate that our algorithm can successfully detect dishonesty. Finally, we show by means of experimental evaluation that Jiminy can scale on-demand to accommodate increasing numbers of participants and subjects.

4.1 Analysis of the data set

nose-length distribution. The ability to judge, given a set of participant ratings, whether a participant is likely to be honest is a crucial element of the system. In [12] we simulated the behavior of *nose-length* values for different users, for various subjects. We predicted that nose-length would have the form of a Gaussian random variable, with a small number of potentially dishonest users being at the far left or right parts of the distribution.

We analysed the GroupLens movie ratings data set to determine the real distribution of nose-length values inside the set, for the users it contains. The nose-length value was plotted against its frequency of occurrence. The result of this analysis is shown in Figure 2; the nose-length distribution does indeed fit the theoretically anticipated Gaussian distribution. This provides a strong indication about a relationship between one’s relative frequency of disagreement and his or her probability of being honest. Section 4.2 demonstrates that this relationship holds, and that Jiminy is able to exploit it to detect dishonesty.

Distribution and correlation of ratings. As expected, our analysis of the chosen data set revealed that ratings given by users to films did not always have a normal distribution. Figure 3(a) shows three density plots of ratings for three different movies namely, “Toy Story”, “Jumanji” and “Big Bully”.

Film ratings are highly subjective. Some participants are likely to be very impressed by a film while others may consider it disappointing. This can lead to ratings exhibiting a multi-modal distribution — for example, approximately

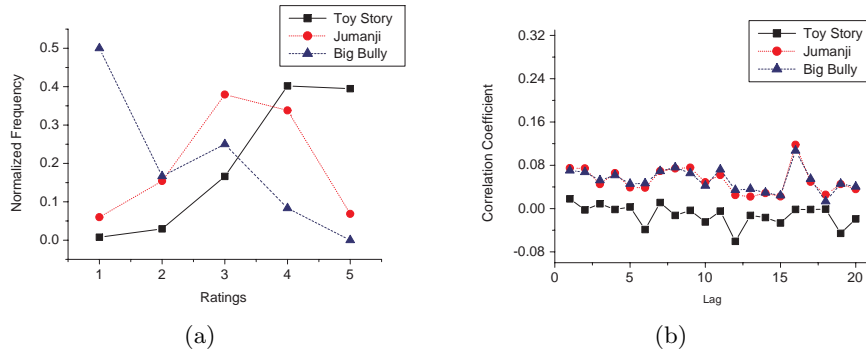


Fig. 3. a) Variation in distributions of ratings, and b) correlation among ratings for three subjects: Toy Story, Jumanji and Big Bully.

half of the participants may assign a rating of 1 or 2, and the other half a rating of 4 or 5. This type of distribution could lead to a mean value which almost no one has entered, and to a high standard deviation for ratings. Our analysis showed that this potential problem does not appear to be severe; most films did have a firm most common rating, although this value may not always be exactly reflected on the mean.

In addition to the distribution of ratings, the correlation of ratings in the GroupLens set was also studied, as illustrated in Figure 3(b). Since the correlation coefficients are very low, it can be safely assumed that the ratings provided by a user are independent of the existing ratings, thus making the rating process independent and identically distributed. This observation emphasises that \bar{T} and $\hat{\sigma}$ of the distributions are appropriate to capture and characterise user honesty.

4.2 Evaluation of the algorithm

To evaluate the effectiveness of Jiminy with respect to assessing the honesty of participants we conducted the following experiment. We injected the ratings of four known dishonest users into the existing GroupLens data set, fed the resulting data set into the Jiminy cluster, and monitored the nose-length values that Jiminy assigned to the known dishonest users.

We created the following four users and subsequently injected their ratings into the data set:

- *Mr. Average.* This user periodically queries the RMS to obtain the average rating for each movie he wishes to rate, and subsequently submits the integer rating closest in value to the average rating reported for the same movie. This average rating reported is unlikely to continually be the most popular rating because of the nature of the ratings' distributions.

<i>User</i>	T_u	Z
Mr. Average	-8.95	19.28
Ms. Popular	-7.84	24.78
Mr. Disagree	-39.35	-132.85
Ms. Random	-20.31	-37.64

Table 1. T_u and nose-length values (x10) of the four users in question

- *Ms. Popular.* This user periodically queries the RMS to establish the most popular rating for each movie she wishes to rate, which she then submits for the same movie.
- *Mr. Disagree.* This user periodically queries the RMS to obtain the average rating for each movie he wishes to rate, and then reports a rating that is as far from the average value as possible. For instance, he would report 1 if the average rating was 5 and vice versa, and he would report 1 or 5 (at random) if the average rating was 3.
- *Ms. Random.* This user periodically submits a random rating for each movie she wishes to rate.

We selected a subset of 43 films from the RMS for these dishonest users to rate, entered their corresponding ratings — one per movie per user, and used Jiminy to assess their honesty values. The results of this experiment are shown in Table 1.

The above shows that dishonest users do have a nose-length (Z value) quite different from that of the honest users. Mr. Average and Ms. Popular have Z values appearing at the far right side of the graph in Figure 2, as predicted. Mr. Disagree and Ms. Random both appear to the left side, with Mr. Disagree off the side of the graph. Interestingly, no users in the original data set disagreed to this extent, but at the opposite end of the scale there were users who agreed with a frequency similar to — though not quite as high as — that of Ms. Popular.

This result demonstrates that the honesty metric is effective, being able to spot our simulated dishonest users in a large data set of real ratings. It can also be used to choose appropriate honesty and dishonesty threshold values, which is discussed in the following section.

Discussion

Selection of threshold values. Choosing honesty and dishonesty thresholds presents a trade-off: setting the dishonesty threshold too high may allow dishonest participants to be rewarded, while setting it too low may punish honest participants who, owing to the subjective nature of the rating topic, vary in opinion a little too frequently or too infrequently. At the same time, setting the honesty threshold too low would make it difficult for a dishonest user to be deemed honest again, while setting it too high would increase fluctuation between the honest and dishonest states.

Jiminy allows for these parameters to be adjusted by the system administrator. Suitable honesty and dishonesty thresholds can be devised through inspection of the nose-lengths of known dishonest users (such as the ones in the previous section), the distribution of nose-lengths, and depending on the trustworthiness of the environment in which the system is deployed. Tuning the thresholds effectively determines the *tolerance* (or harshness) of Jiminy.

As an example, as Figure 2 shows, 89.6 % of participants are within the Z (x10) range -14.5 to 14.5, and 93.34% are within the Z range -17 to 17. Setting the honesty threshold at 14.5 and the dishonesty threshold at 17 would deem 6.66% of participants dishonest.

Rating engineering. Let us consider a participant that submits a number of honest ratings, enough to take her well above the dishonesty threshold. She then submits a mixture of dishonest and honest ratings in varying proportions, and tests whether she is still deemed honest. She keeps increasing the proportion of dishonest ratings until she is deemed dishonest, and then reverses that trend. At some point, the user may find an equilibrium where she can be partially dishonest — but not enough for the system to halt her rewards. We term this type of attack *rating engineering*.

Jiminy has a number of countermeasures against such attacks. First, it does not make the threshold values publicly accessible. At the same time, it conceals fine-grained nose-length values, providing only a binary honest/dishonest answer when queried about a certain user. Additionally, the exponentially increasing probationary period introduces a high cost for such attacks. As credits cannot be traded for money, we believe that the incentive for determined rating engineering is reasonably low. However, as part of our future work we plan to investigate the design of an algorithm with more explicit defences against such attacks.

4.3 Scalability

Experimental setup. To assess the performance and scalability of Jiminy we deployed the system on a cluster composed of five machines, as shown in Table 2. We deployed five slave instances, one on each machine. The master node was run on machine number one, along with one slave instance. The *base value* shown in the table represents the time needed by each machine (running the entire Jiminy system on its own) to update the nose-length of each participant in the GroupLens data set, when 5000 new ratings have been submitted. We term *new ratings* the ones submitted after the latest periodic execution of the algorithm.

The performance difference between the slaves, as indicated by the disparity of base time values, is due to both hardware differences and level of load. For instance, slave number four has been relatively heavily used by third-party applications during the time the experiments were undertaken.

Results. We measured the time needed by slave number one to finish the periodic calculation of nose-lengths for 5000, 10000, 20000, 40000, and 80000 new ratings, and while running in a cluster of one to five slaves. The results of this are

<i>Machine</i>	<i>Specs</i>	<i>base time (s)</i>
1	AMD Opteron 244, 1.8GHz, 2GB RAM	765.85
2	AMD Opteron 244, 1.8GHz, 2GB RAM	764.90
3	UltraSPARC-IIIi, 1.0GHz, 2GB RAM	1904.79
4	Intel Xeon, 3.06 GHz, 1GB RAM	2556.06
5	UltraSPARC-IIIi, 1.0GHz, 8GB RAM	1793.37

Table 2. Specifications and base time for machines in the Jiminy cluster.

shown in Figure 4(a). We observe that the time required increases linearly with the number of new ratings, and that adding slaves to the cluster significantly improves system performance. As an example, for 5000 ratings, slave number one completed its calculation in 161 seconds when five slaves were present, compared to 765 seconds when running on its own. For 20000 ratings the same computation took 676 seconds in a cluster of five and 3110 seconds on slave number one alone.

We also measured the *speedup*, denoted as the ratio of time required when a slave ran alone over the respective time when N slaves were participating in the cluster, to perform the same calculation. We measured the speedup achieved by each slave as new slaves were added to the cluster for 5000, 10000, 20000, 40000, and 80000 new ratings. The results of this experiment for 5000 ratings are shown in Figure 4(b). Speedup results for experiments with more ratings look nearly identical to these. As shown in the graph, each slave achieves a near-linear performance improvement for every new slave that is added to the cluster. This underlines that the cluster can be scaled *on-demand* to accommodate increasing numbers of participants and subjects. The small deviation from a precise linear performance increase is attributed to our user space partitioning scheme, assigning slightly different parts of the user space to slaves.

Also, it is worth noting that the master node was not a bottleneck in the system, presenting a very low CPU and memory utilisation compared to the slaves. Additionally, our experiment demonstrates that the performance and scalability of Jiminy allow it to be deployed in a realistic setting. By increasing its performance linearly when slave machines are added to the cluster, Jiminy can scale to calculate nose-length values at a rate equal to (or higher than) the rate at which ratings are submitted. Our modest five-machine cluster — a commercial deployment would be equipped with more and more capable high-end servers — can process approximately 31 new ratings per second, in a data set of one million ratings in total.

5 Related work

We identify two areas of research as relevant to the work presented in this paper: reputation management systems, and incentive schemes.

Reputation management systems. RMSs are widely employed within a range of systems and communities. In peer-to-peer systems [2, 15] they serve as a means of

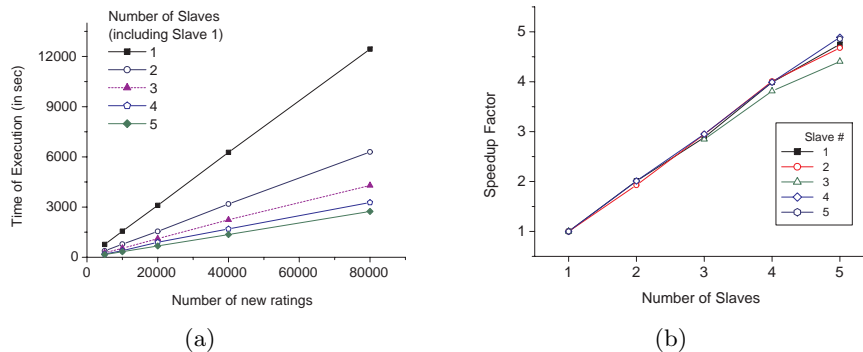


Fig. 4. a) Time taken by the periodic nose-length calculation on slave 1 for different numbers of ratings and slaves present, and b) speedup observed by increasing the number of slaves.

reducing free-riding [3, 14] and selecting reliable peers. In ad-hoc networks, they support detecting selfish nodes [6]. In auction sites⁵ and on-line marketplaces⁶[7] they help buyers identify reliable sellers and vice versa.

We investigated the use of an RMS in public computing [10], and observed that requirements are different in this environment [12]. To summarise, computing resources are regarded as utility and users take good service for granted, thus ratings would tend to only be exceptionally negative and (less frequently so) exceptionally positive. Additionally, interactions happen more frequently and in short timescales, increasing the overall overhead of submitting ratings.

Another observation we make is that there is a trade-off between the *level of anonymity* of participants and their *willingness to participate*. Anonymity and the lack of a sense of permanent identity act as disincentives for active participation. This is part of the reason why files in file-sharing peer-to-peer systems are rarely rated, as well as why peers are indifferent about their reputation — they can easily escape bad ratings by re-registering using a different identity [9].

On the other hand, participants in auction sites care about their reputations more than peers in peer-to-peer systems, due to the semi-permanent, pseudonymous nature of their identities. However, semi-permanent identities provide an incentive for submitting biased information; submitting positive feedback about others is often related to an expectation of reciprocity, while submitting negative feedback is often due to a feeling of revenge [1].

Jiminy complements RMSs, facilitating explicit participation rewards and providing information about the honesty of users. Its XML interface supports interoperability with a variety of RMSs.

⁵ <http://www.ebay.com>, <http://auctions.yahoo.com/>

⁶ <http://www.amazon.co.uk/exec/obidos/tg/stores/static/-/marketplace/welcome/>

Incentive schemes. Providing incentives for participation is a fairly general research avenue. Recent studies have focused on incentives for cooperation between nodes in wireless ad hoc networks [8]. Peer-to-peer systems such as Kazaa⁷ often provide incentives for providing content by linking peer ratings to download rates [4, 18, 19]. Bittorrent⁸ uses a ‘tit-for-tat’ approach to ensure peer participation.

Social means for improving collaboration within communities have been examined [5, 11, 16], suggesting that interactions should carry a notion of *visibility*, *uniqueness*, and *benefit* for the participant. In accordance to these findings, we believe that the explicit incentives that Jiminy provides can help enhance the feeling of a participant that she benefits from submitting ratings.

While incentive models have been proposed before, to the best of our knowledge there is a need for an architecture that a) provides an honesty metric to improve the quality of received feedback, and b) performs and scales well in order to accommodate realistically large data sets.

6 Conclusion

This paper presented the design, implementation, and cluster deployment of Jiminy, a distributed architecture for providing explicit incentives for participation in reputation systems. Jiminy features a reward mechanism and a probabilistic honesty metric based on the Pinocchio [12] model to encourage the submission of honest ratings.

The GroupLens data set of one million real movie ratings was employed to verify our assumptions about the distribution of nose-length values (our honesty estimator), and about the independence of ratings. Furthermore, the system’s effectiveness was evaluated by showing that it is able to detect four typical types of dishonest participants whose ratings we injected into the GroupLens data set. Finally, experiments were conducted to demonstrate that Jiminy performs and scales well, increasing its performance near-linearly as slaves are added to the cluster, and being able to process ratings at run-time.

At the time of writing, we are evaluating Jiminy using different types of data sets, where rating subjectivity is higher or lower. We also plan to investigate algorithms with inherent resilience towards rating engineering, and mechanisms for the dynamic repartitioning of computation according to the performance and workload of individual slaves.

Acknowledgments

We would like to thank Prof. Sahin Albayrak, Silvan Kaiser, and the Technical University of Berlin for providing support for our experiments. We would also like to thank the anonymous reviewers for their constructive comments and suggestions.

⁷ <http://www.kazaa.com>

⁸ <http://bitconjurer.org/BitTorrent>

References

1. A. Abdul-Rahman and S. Hailes. Supporting Trust in Virtual Communities. In *Proc. Hawaii Intl Conf. on Sys. Sciences (HICSS)*, January 2000.
2. K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proc. 10th Intl Conf. of Inf. and Knowledge Mgmt*, 2001.
3. E. Adar and B. Huberman. Free riding on gnutella. Technical report, Xerox PARC, Aug. 2000.
4. K. Anagnostakis and M. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *Proc. 24th Intl Conf. on Dist. Comp. Sys. (ICDCS 2004)*.
5. G. Beenen, K. Ling, X. Wang, K. Chang, D. Frankowski, P. Resnick, and R. E. Kraut. Using social psychology to motivate contributions to online communities. In *Proc. ACM Conf. on Comp. supported Coop. work (CSCW '04)*, 2004.
6. S. Buchegger and J. Y. L. Boudec. A robust reputation system for p2p and mobile ad-hoc networks. In *Proc. 2nd Workshop on the Econ. of Peer-to-Peer Sys. (P2PEcon 2004)*, 2004.
7. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. 1st Intl Conf. on the Practical App. of Int. Agents and Multi-Agent Tech. (PAAM'96)*. Practical Application Company, 1996.
8. J. Crowcroft, R. Gibbens, F. Kelly, and S. Ostring. Modelling incentives for collaboration in mobile ad hoc networks. In *Proc. of WiOpt'03*, 2003.
9. J. R. Douceur. The Sybil Attack. In *Proc. of the 1st Intl Workshop on Peer-to-Peer Sys.*, March 2002.
10. B. Dragovic, S. Hand, T. Harris, E. Kotsovinos, and A. Twigg. Managing trust and reputation in the XenoServer Open Platform. In *Proc. 1st Intl Conf. on Trust Mgmt*, May 2003.
11. T. Erickson, C. Halverson, W. A. Kellogg, M. Laff, and T. Wolf. Social translucence: designing social infrastructures that make collective activity visible. *Communications of the ACM*, 45(4):40–44, 2002.
12. A. Fernandes, E. Kotsovinos, S. Ostring, and B. Dragovic. Pinocchio: Incentives for honest participation in distributed trust management. In *Proc. 2nd Intl Conf. on Trust Management (iTrust 2004)*, Mar. 2004.
13. E. Kotsovinos and A. Williams. BambooTrust: Practical scalable trust management for global public computing. In *Proc. of the 21st Annual ACM Symp. on App. Comp. (SAC)*, Apr. 2006.
14. R. Krishnan, M. D. Smith, and R. Telang. The economics of peer-to-peer networks. Draft technical document, Carnegie Mellon University, 2002.
15. S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative Peer Groups in NICE. In *Proc. of IEEE INFOCOM*, 2003.
16. P. J. Ludford, D. Cosley, D. Frankowski, and L. Terveen. Think different: increasing online community participation using uniqueness and group dissimilarity. In *Proc. SIGCHI Conf. on Human Factors in Comp. Sys. (CHI '04)*, 2004.
17. P. Resnick and R. Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. In *The Econ. Internet and E-Comm.*, volume 11 of *Advances in App. Microec.* Elsevier Science, 2002.
18. Q. Sun and H. Garcia-Molina. Slic: A selfish link based incentive mechanism for unstructured peer-to-peer networks. In *Proc. Intl Conf. on Dist. Comp. Sys. (ICDCS '04)*.
19. B. Yang, T. Condie, S. Kamvar, and H. Garcia-Molina. Non-cooperation in competitive p2p networks. In *Proc. Intl Conf. on Dist. Comp. Sys. (ICDCS '04)*.