A Trust-enabled P2P Recommender System

Georgios Pitsilis, Lindsay Marshall
School of Computing Science, University of Newcastle, UK
{Georgios.Pitsilis,Lindsay.Marshall}@ncl.ac.uk

Abstract

In this paper we present a trust-oriented method that can be used when building P2P recommender systems. We discuss its benefits in comparison to centralized solutions, its requirements, its pitfalls and how these can be overcome. We base the formation of trust on evidential reasoning and designed with ease of adoption by existing infrastructures in mind. The paper includes a first analysis of performance based on a simulation used to investigate the impact on scalability and thus show the applicability of the protocol

Keywords: *P2P systems, Recommender Systems, Trust Modelling, Subjective Logic*

1. Introduction

Recommender systems (RS) are used widely in e-commerce for offering suggestions to customers about products [1]. However, they are not perfect and have faults such as vulnerability to malicious attacks and low quality of predictions caused by sparse datasets. Using *Trust* in recommender systems has a positive effect on quality of service [3,5], but as trust protocols use extra resources, it is not clear if their use is appropriate.

The work in this paper looks at the applicability of a trust-enabled recommender system to a distributed environment, to see if the scalability barriers of centralized approaches can be overcome. We focus on a P2P solution where a virtual Overlay built up from users' relationships supports trust.

2. Motivation

Today's recommender systems operate as centralized services [2,4] and correlate users using opinions they have expressed in the past so as to provide them with suggestions either as product lists or rating predictions.

Trust-enabled recommender systems can give an improvement in the quality of recommendations by decreasing *sparsity*, though if this is done incorrectly the extra effort required may have more negative side-effects than benefits [3]. By *sparsity* we mean a lack of the shared-experience data required for *Collaborative Filtering* systems (CF) to work. The technique requires large amounts of computation to build the trust infrastructure that increases with the number of users. Given the requirement that such systems tend to be interactive, response times must be kept low and any additional computational load may have disastrous consequences for usability. However, P2P systems offer the advantages of being robust, more resistant to attacks and allowing to users to share the cost of the service. A P2P RS would also be able to work unbiased since there is no central source of potentially biased results.

If a system uses a flooding algorithm to propagate trust values to neighbouring users, this would require $O(n^t)$ unicast operations since a query running to depth d would need roughly $L = n\sum_{t=1}^{d} n^{t-1}$ operations for each search operation to propagate to n neighbours. In reality the number of messages will not be as high as indicated due to sparsity in the datasets.

High sparsity also obscures the scalability problem since it is impossible for all users to be correlated with each other. This means that predictions cannot be made about every possible user choice within the community. *Computability* is a measure used to express how well a system can provide predictions about a range of products, and, as [3] shows, the deployment of trust in a CF system helps to increase *Computability* without significant impact on *Prediction Error* in comparison to a plain CF system.

The two challenges (*reduction of sparseness* and *scalability)* conflict, since the less time spent on a search query, the worse the results. The increasing number of computations is the main reason for reduced scalability of CF systems and is why, theoretically, a centralized recommender system of this type cannot scale to large number of users and products.
The contribution of our paper is two-fold:

- To describe a protocol that can use trust in a P2P recommendation system

- To investigate the scalability of such a protocol by providing simulation results.

3 Related work

Recommender systems [20] generally work as centralized services and often exist as services embedded into web sites which provide support for e-commerce activities. *epinions.com* [2], *amazon.com* [16] and *eBay* [4] are examples of popular sites that provide recommendations. The idea behind CF is to predict scores based on the heuristic that people who agreed (or disagreed) in the past will probably agree (or disagree) again.

Proposals for interesting designs of distributed P2P recommendation systems, such as the work of Kinateder et.al. [10], have been made in the past, but no performance measures have been presented to show the limitations of such solutions.

In the area of P2P systems simulation, K.Kant [14] has carried out performance evaluation for resource sharing networks using analytic modeling. Also, work done by B.Yang and H.Garcia-Molina [15] provides evaluations of a wide range of configurations of P2P networks based on existing file sharing protocols. Even though these papers are good references for sizing P2P networks, they are restricted only to resource location operations and not establishment and discovery of trust.

We should also mention the work done by K.Aberer and Z.Despotovic [21] on trust-management, but it does not solve the sparsity problems that recommendation systems have when there are insufficient data to support a recommendation.

E.Damiani et.al. [22] proposed a reputation mechanism, tailored to the Gnutella algorithm, that makes reliability checks on candidate participants prior to downloading by sending polls to all neighbouring peers via a reputation exchange protocol which they call XRep. The purpose of this protocol is to collect votes from those peers that have past experience with the peer under examination. After all the information has been collected, it is validated by sending another poll, and, finally, used by the querying peer to reach to a decision. Peers maintain local repositories of opinions based on subjective criteria about others with which they have had transactions in the past. Even though there are no evaluation results or performance analysis for the improvements that the algorithm offers for retrieval operations, the polling itself affects the scalability of the Gnutella protocol due to the extra messages that need to be sent out and roughly triples the traffic.

4 Our Proposed Architecture

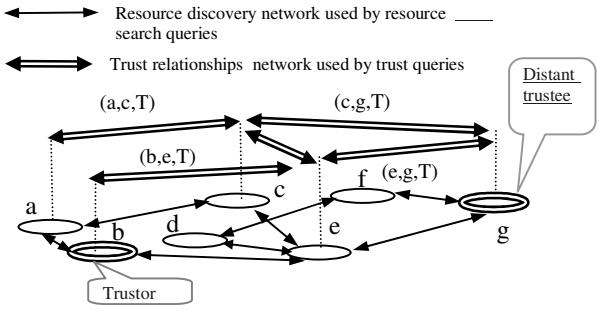

Figure 1. A typical trust Overlay. (a,c,T) represents the trust of a for c.

Our categorization of phases in recommendation production is similar to that of Sarwar et. al. for Recommender Systems [6] and in our P2P based system we distinguish 3 key phases: *Trust discovery*, *Recommendation Search* and *Recommendation Generation*. In order to explain these operations we first present a common scenario for searching in a P2P system that provides trust-based recommendations:

1. User U initiates a query searching for some product A she is interested in.
2. The system detects, through the usual resource discovery operations, the subset S of entities that can provide their experiences with the product A.
3. The system tries to estimate the trustworthiness (*Secondary Trust*) of the S entities through the trust graph and informs U.
4. Once the trust information has been received by U and the trust for each S has been established, U derives the expected rating of product A using the ratings of each of the S entities

Figure 1 shows an example of two entities **g** and **b** which share some experience with product A. In the scenario, entity **a** is also interested in product A and is trying to derive the trustworthiness of **g** and **b** (both untrusted neighbours of **a)** through the trust graph.

Trust establishment is the operation of estimating the levels of belief between users using their behavioural data as evidence. *Subjective logic* is a framework of artificial reasoning suitable for modelling such relationships which conveniently also deals with the fact that knowledge is always imperfect [9]. In this framework the ratio of trust, distrust and absence of data are expressed as triplets of belief, disbelief and uncertainty (b,d,u). *Subjective Logic* provides solutions to matters that have to do with trust propagation or transitivity which is a basic part of our technique.

4.1 Trust Discovery

The *Trust Discovery* phase is concerned with the representation of trust relationships and the formation of neighbourhoods of users based on their level of mutual trust. This is known as *Primary Trust*, and this phase must take place before a trust query starts. Every entity that wants to take part in the scheme must implement it. In contrast to centralized *recommender systems*, users maintain their own tables for chosing neighbours. [7] contains an empirical model for trust derivation using evidence in which trust values derive from the shared experiences of pairs of users. It provides a mapping that uses both quantitative and qualitative measures to transform ratings into opinions. The more easily two parties can predict the ratings of their common choices, the more they trusted each other. Trust values take the form of triplets (b,d,u) so they can be used with subjective logic algebra.

The operations of the Trust Derivation phase can are carried out off-line by the parties involved which can periodically broadcast messages indicating that they are looking for others to establish primary trust with.

Once the *Primary Trust* relationships have been established the trust Overlay is set and ready to resolve trust requests. To keep the protocol simple we assume everyone is honest and allow calculations to be done by the entities themselves.

4.2 Recommendation Search

Once the trust graph has been built, *Recommendation Search* queries can be serviced. The purpose of these queries is to derive the trustworthiness of an entity (Destination) that is known to have useful experiences, from the point of view of another entity (Origin). We assume that a flooding scheme to propagate trust queries to a defined hop distance.

In order to shape the trust paths from the origin to the destination the intermediate trust vectors must be sent back to the origin once the query has reached its destination. There is also a requirement for *common purpose* [13] to exist in the relationships to make it possible for users to use transitive trust via the graph.

Upon receipt of all replies the origin then maintains a collection of trust vectors forming a graph leading to the destination. It then analyses the resulting graph so as to calculate the target node's *derived trust*. Various quality restrictions can be set when traversing the graph, such as avoiding entities with low numbers of experiences or those that are untrustworthy. This reduces the number of unimportant links in the resulting graph and thus reduces the trust calculation time. [3] shows that using filters in trust queries has no serious impact on the error of the predicted recommendations. As the search goes deeper, the number of trusted entities reached rises. Due to the use of flooding protocols this increases the number of messages needed exponentially and affects scalability.

The reason the resource hungry task of parsing the trust graph is done by the entity itself and not step-by-step by the intermediate entities is that this preserves the dependency avoidance requirement of trust. Doing otherwise may lead to an incorrect calculation of *Derived Trust* due to hidden topologies of which the originator is not aware. This is the *Dependency Problem* in trust derivation [8].

4.3 Recommendation Generation

Finally, *Recommendation Generation* consists of turning all *Derived Trust* query results from the previous step into similarity measures for the entities that provided recommendations about the destination. This is the same process as can be found in a plain CF. For example, *Grouplens* uses Resnick's formula [10] which predicts the rating that a user would give an item.

5 Testing

As we mentioned in the previous paragraph, the scalability problems of a trust-enabled recommender system come mainly from the requirement of *Subjective Logic* for opinion independence, which can be fulfilled by transmitting a whole graph structure back to the querying entity. The impact on scalability is two fold: First, the network traffic caused by the number of trust vectors that go through the connections and second, the system overhead for parsing the received structure.

5.1 The protocol explained.

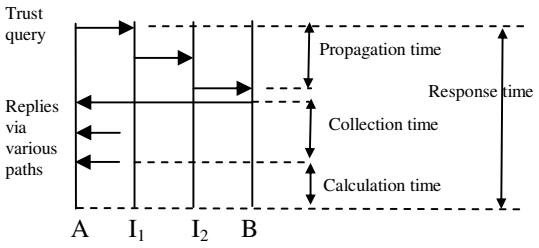

Fig. 2. The sequence diagram for the trust query and the collection of opinions

The trustor *A* initiates queries through its neighbours I1 which forward the message to their neighbours I2 and so on until they reach (the trustee) B. Once the message has reached its destination a reply that contains the whole path travelled ($A \to I_1, I_1 \to I_2, I_2 \to B$) is

sent back to the originator (A) directly. Thus *A* gets informed of what is between itself and *B*. Figure 2 shows the communication operations that take place in a typical case. Fig. 3 shows a simplified recursive form of the algorithm for the collection of the trust vectors between the originators and the destinations used in our simulation model.

```
DEFINE Origin = The trustor Entity
DEFINE Target = The trustee Entity
DEFINE Depth = The depth in the recursive search
DEFINE TrustFilter = The minimum trust value in a trust relation
to be  called the algorithm recursively

Procedure Find_path(Origin,Target,Depth,TrustFilter)

  If Depth=0 then Return(false)
    For all Trust Vectors with Vector.origin=Origin and
                               Vector.bdu>TrustFilter  do
       Read next Trust Vector(user1,user2,bdu)
       Indicator=false
       If user2=Target Then
          SaveVector(user1,user2,bdu)
          Indicator=true
       Else
          If Find_path(user2,Target, Depth-1,TrustFilter)=true Then
             SaveVector(user1,user2,bdu)
             Indicator=true
          Endif
       Endif
    End_for
    If (Indicator=true) then Return(true)
    Else                     Return(false)
End
```

Figure 3.

The *SaveVector* command is used to build the result-set in which the entire sub-graph from Origin to Destination is saved. When the algorithm ends, the result-set contains all the successful paths. In our experiment we used data taken from the MovieLens film recommender database to generate graphs of primary trust vectors. The trust vectors were created from these recommendations as described in 4.1

5.2 Assumptions

To perform our tests we made some assumptions about the conditions under which operations take place.

- We assume the pattern of user demand follows a *Poisson* pdf with $\lambda=1(\text{min}^{-1})$ i.e. each minute the user submits on average 1 request to the system.
- Every trust vector in the graph is stored as a triplet *{Primary Trust value(b,d,u), Trustor, Trustee}* and has a fixed size of 50 bytes : 15 bytes for the trust value, 10 bytes split into the addresses of *trustor* and *trustee* respectively and the remainder for TCP headers.
- Calculating derived trust for a destination is done by applying the consensus and discounting operators of subjective logic [8], to simplify parallel and serial combinations of opinion vectors respectively. The time required for a simplification operation depends on the number of vectors in the graph. Assuming that an elementary simplification operation in Java takes about 1 msec, we conclude for the sake of the experiment that in all it takes:

$$t_s = \frac{1}{1000}\text{sec} \times Size\_of\_Graph$$

- The time required for the collection of the depends on the available bandwidth and is calculated as:

$$t_c = \frac{Size\_of\_\text{Re}plying\_Graph \times V}{Available\_Banwidth}$$

where V is the size of the trust vector in bytes. We assume that V = 25.

- Trust graph analysis begins when the query originator receives all the reply vectors or the patience limit has expired (whichever comes first)
- In the test peers reply with absolute honesty.

5.3 Variables

We identified 3 variables that generate 60 different testing scenarios. These are:

- Trust filter. This represents the minimum level of belief that a trustor must place on some neighbouring trustee in order to allow a query to propagate. We used 3 filters b>0.5 , b>0.6 , and b>0.7.
- The bandwidth of the network connection that links the node to the network. In total we performed tests for speeds of 7 kb/sec and 64 kb/sec, simulating an analog modem and a DSL connection respectively.
- Community size ranging from 5 to 100 users.

5.4 Test Plan

For all combinations of *propagation filter*, *community size* and *bandwidth capacity,* we ran 10 sessions of 2 hours of simulated time and averaged the results. 2 Hours was chosen so that the system reached a steady state before we took measurements.

To evaluate the model we used *Response time,* and *Success Rate*. The first expresses the time between query initialisation and completion of the derived trust calculation. To find this, we ran queries for each node and measured the propagation delay to B, the waiting time for all responses and the calculation time for the secondary trust of A for B. The sum of the these gives *Response time*. We introduced *Success rate* because pure response time did not have a significant value. *Success Rate* expresses the probability that a query completes within a threshold of 10sec. We chose this as a reasonable value and we aimed to see how many times this value was exceeded. This threshold represents a

measure called the *Patience limit* and studies in interactive environments [11] show for most users *Patience limit* has this value. If there is no response in the time, users may abandon the request or retry. Abandonment in P2P environments is expensive in resources because it adds load and worsens the situation [12]. We treat both abandonment and retry as unsuccessful queries.

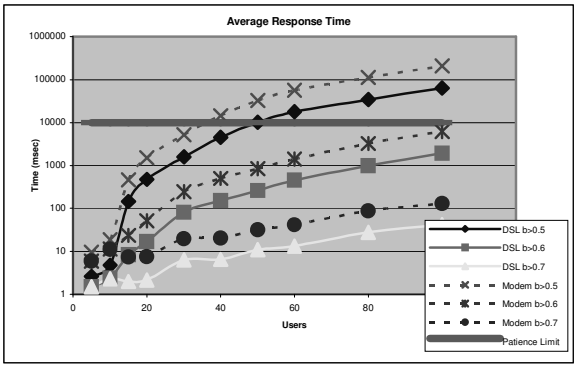

Fig4. Expected Response times for various communities

5.5 Results

Figure 4 shows the Expected response time of a query for each of the 60 configurations tested for various community sizes with a maximum hop distance of 3.

The line showing the Patience limit is also drawn. For comparison, we tried three different trust propagation filters (b>0.5, b>0.6, b>0.7). There are cases where response times are within the limit especially when we apply strong filtering.

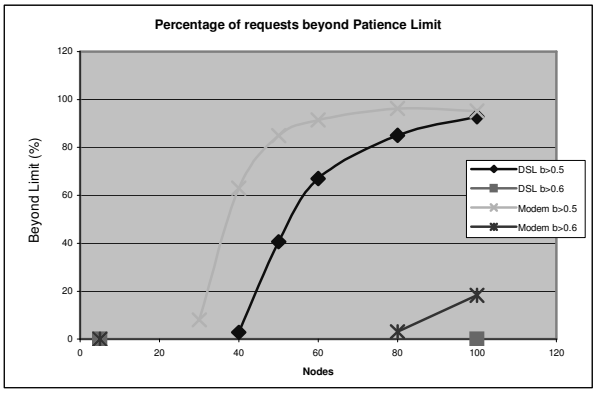

Fig 5. Responses beyond Patience Limit for various filters

We also measured the percentage of queries whose response times exceeded the threshold and so were considered unsuccessful (fig. 5). The complement of this measure is *Success Rate*. We present results for nodes using DSL connections and modems, and do not show data for trust filter b>0.7 because there were no unsuccessful queries with up to 100 peers.

The notion of *Success Rate* does not seem to be enough to find the best choice because it does not include a metric of applicability for each individual case along with the Success. We therefore introduced *Satisfaction factor* (*SF*) to give a measure based on both Success Rate and the coverage achieved by using a filtering policy. We define:

***Satisfaction Factor* = SuccessRate * NCoverage** (1)

Coverage factor (NCoverage) is the percentage of services for which a user can find opinions through the trust graph in relation to the total number of services rated by all peers. NCoverage also takes care of the fact that there is always prediction error. A definition and more about the *NCoverage* can be found in [3].

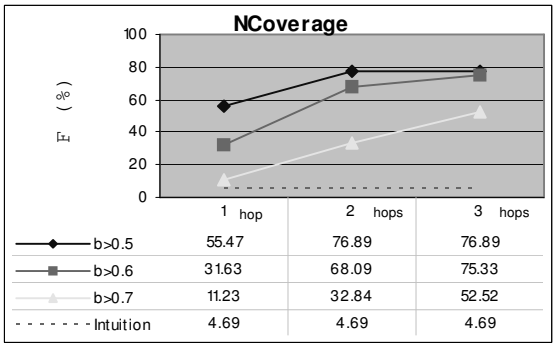

Figure. 6. Normalized Coverage factor

Fig. 6 shows the values that *NCoverage* takes for 3 scenarios and for various message propagation hop. These values were derived from an experiment in [3]. Here we are interested only in values of NCoverage for 3 hops. *Success Rate* can be taken from the complement of the values presented in Figure 5.

Trust Filter	b>0.5	b>0.6	b>0.7
N-Coverage	*76.89*	*75.33*	*53.53*
100 DSL	3.80	61.5	53.53
100 Modem	5.67	71.56	53.53
80 DSL	3.01	72.99	53.53
80 Modem	11.9	75.33	53.53

Figure 7. User Satisfaction for various trust filters

Figure 7 displays the values of NCoverage along with the *Satisfaction Factor* for two different network configurations and for various trust filters and sizes of communities (80 and 100 peers). By trust filter we mean the filter applied for the propagation of a trust query from one node to a neighbouring one in the way that is being used in the algorithm of Fig. 3. For example a filter b>0.5 would not allow a trust query to propagate to a linked neighbour which is not trusted at least as 0.5. This filter impacts the density of the trust graph.

From the *Satisfaction Factor* can be seen that DSL users get the highest satisfaction, for both sizes of communities, when the middle trust propagation policy is applied. The strong filtering policy (b>0.7) is not

doing well because of its low coverage no matter the size of the community.

Another interesting characteristic is that in both DSL and modem connections the weak filtering policy for trust (b>0.5) gives almost no satisfaction to the users. This happens because there is high congestion due to the high density in the trust graph and as a result the responses are received outside the 10 sec threshold we have set (Patience limit). From all combinations tested the best satisfaction is received when the middle trust filtering policy is applied. It can be seen from the numbers that the highest value is achieved when the trust filter has been set to 0.6 no matter the speed of connection.

As regards the size of the communities, the numbers show that the satisfaction declines as the communities grow.

Figure 4 shows that there is no significant difference in the Response time between modem and DSL. As regards how the system scales, it shows that as the number of peers grows, the response times increase exponentially and this imposes a scalability problem. Due to lack of resources we were unable to show what trend the Response Time and the Success Rate follows when the number of users exceeds the 100.

6 Conclusion - Future Work

The benefits of decentralized architectures, such as P2P, seem to be suitable for supporting traditionally centralized services such as Recommender Systems. In this paper we presented the idea of using a trust-enabled RS over a P2P infrastructure. The simple analysis we performed based on real data shows that, within the limitations of the current technological infrastructure, such an architecture can favourably support a mid-scale, distributed, trust-enabled Recommender system. In comparison with centralized approaches the distributed recommendation system we propose has the advantage of being robust i.e. recommendation provision is not disrupted even when some nodes in the system collapse.

That we have run the tests for a relatively small community, is of no significance as there is no need to employ more than this number to derive a trust recommendation. Even if a P2P system has a large number of peers, those most appropriate for building the trust overlay could be selected by using an appropriate trust propagation filter. The analysis done in [3] shows that *Prediction error* is not affected seriously by the filters used in trust propagation. Therefore we suggest that filters should be used in cases where the response times cannot be kept within the acceptable limits.

The purpose of our study was to provide a macroscopic analysis of how a flooding protocol behaves for the trust propagation as the user community grows, considering the effects of its operation on the rest of the network infrastructure and on the peers themselves. Our future plan is to build an analytic model that will help us to study these issues from a macroscopic view and which would also give an understanding of the protocol performance in extreme situations that we have been unable to test due to lack of experimental data (e.g. propagating trust beyond 3 hops distances). Comparison with an identical centralized solution with regard to the cost/value ratio is an important future issue.

References

[1] P.Resnick, H.R.Varian, "Recommender Systems", Communications of the ACM, 40(3): 56-58, 1997

[2] http://www.epinions.com

[3] G.Pitsilis, L.F. Marshall, "Trust as a key to improving recommendation systems", in Proc of Third International Conference iTrust 2005, Paris France, May 2005.

[4] http://www.ebay.com

[5] P.Massa, P.Avesani, "Trust-aware Collaborative Filtering for recommender Systems", CoopIS/DOA/ODBASE (1) 2004: 492-508

[6] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, "Analysis of Recommendation Algorithms for E-Commerce", In Proceedings of the second ACM conference on Electronic Commerce, pg.158-167,ACM Press, 2000.

[7] G. Pitsilis, L.F. Marshall, "A model of Trust derivation from Evidence for Use in Recommendation systems", In Proc PREP 2005, Presented Poster, Lancaster, April 2005.

[8] A.Jøsang, S. Pope, "Semantic Constraints for Trust Transitivity", Second Asia-Pacific Conference on Conceptual Modeling (APCCM2005), Newcastle, Australia, January-February 2005.

[9] A.Jøsang, "A Logic for Uncertain probabilities", International Journal of Uncertainty, fuzziness and Knowledge based systems, Vol.9,No.3, June 2001.

[10] M.Kinatender, K.Rothermel, "Architecture and Algorithms for a Distributed Reputation System", iTrust'03, Heraclion , Greece, 2003.

[11] R.B.Miller Response time in man-computer conversational transactions. Proc. AFIPS Fall joint Computer Conference Vol. 300,267-277. (1968)

[12] K. Kant , R. Iyer," A performance model for Peer to Peer File Sharing Service", Enterprise Architecture Lab, Intel Corporation, Technical Report, 16 November 2001.

[13] A.Jøsang, E.Gray, M.Kinateder, "Analyzing topologies of Transitive Trust", In proceedings of the Workshop of Formal Aspects of Security and Trust, (FAST 2003), Piza September 2003.

[14] K.Kant, "An Analytic model for Peer-to-Peer file sharing networks", International Communications Conference, 2003.

[15] B.Yang, H.Garcia-Molina, "Designing a Super-Peer network." In Proc. of the19th International Conference on Data Engineering (ICDE), Bangalore, India, 5-8 March 2003, IEEE Computer Society 2003.

[16] http://www.amazon.com