CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **21**:1389–1403
Published online 8 May 2009 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1430

Concurrent access control for multi-user and multi-processor systems based on trust relationships

Isaac Agudo*,†, Carmen Fernandez-Gago and Javier Lopez

Department of Computer Science, University of Malaga, 29071 Malaga, Spain

SUMMARY

Concurrent access control is an old problem in many fields in Computer Science. It has been solved in many languages and systems, using mechanisms like monitors or priority queues. Nowadays computers implement multi-core capabilities. This means that they are virtually capable of execution of processes in parallel. This requires new techniques and open new issues in the field of concurrent access control. Moreover, most operating systems are multi-user; thus, we have to focus on a multi-processor multi-user scenario. Trust becomes a paramount aspect when building distributed applications; the same applies on a lower scale in modern computers. We propose the use of a trust graph that keeps record of the trust relationships of the system and helps in deciding on concurrent access requests. The information encoded in the graph will be used both in order to decide on the access requests and to order granted requests in terms of their associated trust level. Copyright © 2009 John Wiley & Sons, Ltd.

Received 20 November 2008; Revised 29 January 2009; Accepted 12 February 2009

KEY WORDS: access control; trust management; rating; quota

1. INTRODUCTION

Access control is perhaps one of the main concerns within security. Access control is in charge of determining who can gain access or it is authorized to use what resources. Thus, authorization and authentication mechanisms are embedded in the access control process. This implies that certain rules or policies must exist in order to carry out the authorization decisions.

*Correspondence to: Isaac Agudo, Department of Computer Science, University of Malaga, 29071 Malaga, Spain.
†E-mail: isaac@lcc.uma.es

Contract/grant sponsor: European Commission; contract/grant number: FP7-ICT-2007-1-217098
Contract/grant sponsor: Spanish Ministry of Science and Education; contract/grant number: TIN2006-09242

Trust management has been used since its origin as a tool for assisting access control techniques, for instance for defining policies. Thus, the first attempt of designing trust management systems, PolicyMaker [1] and its successors KeyNote [2] or REFEREE [3], were in fact developed as an extra tool for guiding the access control process. The idea behind PolicyMaker was to build a unified common language for policies, credentials and trust relationships expressed as programs.

Since then other works and systems have been developed aiming to solve the problem of access control by using trust management. One of them is the role-based trust management policy language RT [4] designed to support highly decentralized attribute-based access control.

Concurrent access control requires deciding not only whether a request is to be granted but also which one is to be served first.

If we are able to define a trust value for each request, we would be able to order them and serve the most trusted ones at a time. A mechanism for this trust assignment can be based only on the trust relationship between the resource manager and the requester. This way, the trust value associated with the request will be the one associated with the requester.

Different approaches can be followed apart from that. One possibility is by using reputation. In this case we associate a trust value with a request based on the reputation of the requester. Another possibility could use the reliability of the requester.

In some scenarios, where a bunch of processes is running in a computer, and is launched by the same user, we need a finer grained mechanism to make a decision about access control requests and furthermore to order them. Many operating systems support the concept of priority queues [5], which help in deciding on concurrent access control. We can adopt a similar mechanism where each process launched by a user is assigned a rating that determines its priority among the processes of the same user. This rating will be associated with the trust value assigned to the user that launched the process.

Some processes may be sub-processes of others, which means that they were launched in the context of a parent process. This relationship between processes can be also seen as a trust relationship, given that the parent process is delegating in the sub-process execution of a given task. This trust relationship can also be graded using a rating.

This rating may be defined beforehand for each of the sub-processes, but it can also be computed automatically by the system, based on the number of sub-processes launched, given that all the relative ratings sum is less than one. The ratings of the sub-processes will be the multiplication of the relative ratings by the rating of the process that launches it. In the case where there is a sub-process of a sub-process, we would act accordingly.

In order to compare the priority of two processes belonging to different users, the priority ratings have to be multiplied by the trust value associated with the user. If we want to prevent a user with a high trust value from gaining total access to the resources of the system, the priority ratings of all of his direct processes have to sum 1. In this way, not all the processes associated with this user will obtain the highest priority value.

Then we have to solve the problem in two phases. First we compute the trust values of each user in the system. After that, and based on these values, we compute a priority value for each of the processes in such a way that the access request can be served accordingly.

The way trust is computed, i.e. the *trust metrics* used is also a very important issue, which has been widely investigated. The simplest way to define a trust metric is by using a discrete model where an entity can be either ‘trusted’ or ‘not trusted’. More complex metrics use integer or real numbers, logical formulae like BAN logic [6] or vector-like approaches [7]. In the early nineties, the first proposals for trust metrics were developed in order to support Public Key Infrastructure (for instance [8]). In the recent years the development of new networks or systems such as P2P or Ad-Hoc networks, or ubiquitous or mobile computing has led to the imminent growth of the development of trust management systems and consequently metrics for them. Most of the used metrics are based on probabilistic or statistics models (see [9] for a survey on this.). Also due to the growth of online communities, the use of different metrics has become an issue (see for example the reputation scores that eBay uses [10]).

The structure of the paper is as follows. Section 2 shows how we can compute the trust value associated with a user. Section 3 shows how the process of computing ratings can take place and the mathematical model underlying it. The way access control decisions can be made is described in Section 4. Section 4.1 shows an example of the proposed model and the paper is concluded in Section 5 where future work is also outlined.

2. USER TRUST COMPUTATION

Trust relationships can be modelled using a graph where the vertices are identified with the users of the grid and the edges correspond to trust relationships between them. Trust can be defined as the level of confidence that an entity s places on another entity t for performing a given task in a proper and honest way. The confidence level may vary depending on the task. Assuming that the level of confidence is a real number and that for each task there is only one trust value associated in our reasoning system, the trust graph is a weighted digraph. In our scenario, different users perform several requests over the same resource, therefore we only consider just one task.

Once the graph is built we are interested in computing the end-to end trust between two given entities in the graph. A detailed description of how the computation can be performed can be found in [11]. These computations are based on the concept of *trust statements*. Each entity makes trust statements about the rest of the entities, regarding the resource considered for each case. Those trust statements are defined as a tuple (*Trustor*, *Trustee*, *Resource*, *Value*) where we encode both the trustor and the trustee as two entities in the system: the resource for what the trustee is trusted to access and a trust value that is measured in a trust domain. A *trust domain* is in essence an ordered set used in order to measure trust.

2.1. Trust evaluations

If we want to establish trust between two entities in a system, this trust should be measured somehow. A simple way to measure trust could be established by using a binary discrete model where the trust values are set as *a lot of trust*, for a very trusted entity, or *very little trust* if the trust placed in the entity measured is very low. More complicated systems could use integer numbers (Advogato's trust metric or FreeHaven [12]) or real numbers [13,14].

A *trust evaluation* or trust metric is a function such that given a trust graph, G, and two entities s and t, called the source and the target of trust, respectively (trustor and trustee are alternative names for those entities), returns the level of trust or confidence that s places on t.

As the same entities can be trusted in different ways depending on the resource to be accessed, this function also takes into account such a resource as a parameter.

In [11] we focused on trust evaluations that can be decomposed into two elemental functions: the sequential trust function and the parallel trust function. By decomposed functions we mean that the trust evaluation is computed by applying the parallel trust function to the results of applying the sequential trust function over all the paths connecting two given entities.

A sequential trust function is a function that calculates the trust level associated with a path or chain of trust statements, such that the result is zero if and only if one of the statements in the path has a trust value of zero.

The sequential trust function may verify some of the following properties:

- Monotony (Parallel Monotony): Increasing the trust value associated with one of the statements of the path results on a higher trust value for the whole path.
- Minimality: The trust value obtained from several sets of trust values cannot be higher than the minimum of those values.
- Sequential Monotony: If a trust value is added to a path then the resulting trust value should be less than the value associated with the original path.

We can define those functions recursively based on the behaviour over a pair of elements, i.e. we define the operator and apply it recursively.

Definition 2.1 (*Sequential Operator*). A Sequential Operator or Generator Sequential Function is defined as a function $\odot : TD \times TD \longrightarrow TD$ such that $a \odot b = 0$ if and only if $a = 0$ or $b = 0$. $\odot(a, b)$ or $a \odot b$ are used indistinctively for representing the same, whichever is more convenient.

Given a sequential operator, the sequential function can be defined as $f_{\odot}(z_1, \ldots, z_{n-1}, z_n) = f_{\odot}(z_1, \ldots, z_{n-1}) \odot z_n$. Checking the properties above is easier when dealing with the operator. For example, if $a \odot b \leq \min(a, b)$, for any a and b, we could conclude that f verifies the minimality property.

A parallel trust function is used to calculate the trust level associated with a set of paths or chains of trust statements. This function is characterized for behaving as the identity function when there is only one path and by remaining unaltered when null paths are removed from the arguments.

This function may verify the following desirable properties:

- Idempotency. When all the values of the argument paths are the same, then the final value also coincides with this value.
- Monotony. If the trust values associated with one of the argument paths is increased, the result is also increased.
- Associativity. Analogous to the mathematical concept.

The *Generator Parallel Function*, or the *Parallel Operator*, $\oplus$ for the function g, is defined analogously as the operator $\odot$.

Definition 2.2 (*Parallel Operator*). A Parallel Operator or Generator Parallel Function is defined as a function, $\oplus : TD \times TD \longrightarrow TD$, such that $a \oplus 0 = 0 \oplus a = a$.

We say that the two operators $\oplus$ and $\odot$ are distributive if $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$.

In the case where there are no cycles in the trust graph, the set of paths connecting any two given entities is finite. Then, given a sequential operator $\odot$ and a commutative parallel operator $\oplus$, i.e. $a \oplus b = b \oplus a$, the associated trust evaluation, $\hat{\mathscr{F}}_G$, is defined as follows:

Definition 2.3. Let $S_x^{s,t}$ be the set of all paths of trust statements for task x starting in s and ending in t. For each path $p \in S_x^{s,t}$ represented as $s \xrightarrow{v_1} \cdots \xrightarrow{v_n} t$ let z_p be $v_1 \odot \cdots \odot v_n$, then $\hat{\mathscr{F}}_G(s,t,x)$ is defined as $\bigoplus_{p \in S_x^{s,t}} z_p$.

Given a fixed sequential operator, for any parallel operator that verifies idempotency and monotony properties, $z_* = \min_{p \in S_x^{s,t}} z_p \leq \bigoplus_{p \in S_x^{s,t}} z_p \leq \max_{p \in S_x^{s,t}} z_p = z^*$. Therefore, the maximum and minimum possible trust values associated with a path from s to t are the upper and lower bounds for the trust evaluation $\hat{\mathscr{F}}_G$.

Fortunately, we do not need to compute the trust values of each path in order to compute those bounds, i.e. z_* and z^*. In this case we can use an algorithm, adapted from the Dijkstra algorithm [15] to find, for example, the maximum trust path from a fixed entity s to any other entity on the system. The minimum trust path can be computed in an analogous way.

```
foreach $vertex \in G$ do
    $c[vertex] := 0$;
    $prior[vertex] := undefined$;
$trust[s_0] := 1$;
$Q := copy(G)$;
while $Q$ is not empty do
    $a := extract\_max\_trust(Q)$;
    foreach neighbor $b$ of $a$ do
        $alt = trust[a] \odot trust(a,b)$);
        if $alt > trust[b]$ then
            $trust[b] := alt$;
            $prior[b] := a$;
```

Algorithm 1: Algorithm to compute maximal trust paths.

This is a particular case of a trust evaluation where we use the maximum function as a parallel function. Unfortunately, we cannot generalize this kind of algorithm for other combinations of parallel and sequential functions as it heavily relies on the properties of the max and min functions.

2.2. Performing trust computations using matrices

Let us assume the case where there are no cycles in the trust graph. If there are cycles in the trust graph, the following definitions and algorithm are not valid. In fact, the algorithm is different only when we are computing the trust value of a node involved in a cycle. The key aspect of this algorithm is to remove redundant graphs from the trust graph in such a way that the set of paths connecting two given entities remains finite. For a full description of this algorithm see [11].

We could then model the trust network as a triangular matrix, A, where each element a_{ij} represents the trust level that node i places on node j. If we replace the scalar addition and multiplication in matrices by the operators $\oplus$ and $\odot$, respectively, then by iterating powers of the trust network we can compute the node to node trust values of the network. Thus, the trust evaluation could be defined through $\odot$ and $\oplus$ by applying the generalized matrix product algorithm. It is then defined as

$$A \otimes B = \bigoplus_{k=1}^{n} (a_{ik} \odot b_{kj})$$

The generalized product is associative from the left-hand side, i.e. $A \otimes B \otimes C = (A \otimes B) \otimes C$. Therefore, we can define the generalized power of A as

$$A^{(n)} = \bigotimes_{i=1}^{n} A$$

Finally, we can define the operator $\oplus$ over matrices as $A \oplus B = (a_{ij} \oplus b_{ij})$, that can be used as the summation of matrices. Then, given a matrix A of order n the matrix $\hat{A}$ can be defined as

$$\hat{A} = \bigoplus_{k=1}^{n} A^{(k)}$$

$\hat{a}_{ij}$ represents the value of the trust evaluation from i to j.

These definitions become more relevant when the aforementioned functions are distributive and associative. However, they are still valid if these properties do not hold, although they may not be that meaningful.

Next, we will show that under the conditions mentioned above, the element (i, k) in the matrix $\hat{A}$ is the value of the trust evaluation of all the trust paths from i to j. In particular, the first row in this matrix will give us the distribution of trust in the system.

First, we will show that the kth generalized power of the trust matrix $A(A^{(k)})$ contains the trust through paths of length k. We will prove this by induction where the base case holds by the definition of matrix A.

We assume that for $k \in \mathbb{N}$ the element (i, j) in $A^{(k)}$, $a_{ij}^{(k)}$, represents the value of the trust evaluation of all the trust paths of length k from i to j. We will then show that this is also the case for length $k+1$.

Let $C_{ij} = \{c_{ij}^{1}, \ldots, c_{ij}^{m_{ij}}\}$ be the set of all the trust values associated with all the trust paths of length k from i to j, i.e.

$$a_{ij}^{(k)} = \bigoplus_{p=1}^{m_{ij}} c_{ij}^{p}$$

Then since $A^{(k+1)} = A^{(k)} \otimes A$, each element of the matrix can be obtained as

$$a_{ij}^{k+1} = \bigoplus_{l=1}^{n} (a_{il}^{(k)} \odot a_{lj}) = \bigoplus_{l=1}^{n} \left(\left(\bigoplus_{p=1}^{m_{il}} c_{il}^{p} \right) \odot a_{lj} \right) = \bigoplus_{l=1}^{n} \bigoplus_{p=1}^{m_{il}} (c_{il}^{p} \odot a_{lj})$$

Let $c \equiv i \xrightarrow{z_1} c_1 \xrightarrow{z_2} \cdots c_n \xrightarrow{z_k} l \xrightarrow{z_{k+1}} j$ be a $k+1$ length path from i to j now. This path can also be expressed as the path c' of length k from i to l linked with the path from l to j, (l, j).

Therefore, any path of length $k+1$ from i to j can be obtained by adding a link to any path of length k. Thus, a_{ij}^{k+1} represents the value of the trust evaluation of all the paths of length $k+1$ from i to j.

2.3. Efficiency analysis of trust computations

The consecutive powers of matrix A have mainly zeros as their elements. Thus, for example, while A is an upper triangular matrix, the elements of the diagonal a_{ii+1}^2 of the matrix $A^2=(a_{i\,j}^2)$ are all 0. In addition, if the size of the matrix A is n then A^{n-1} has only one element that is not 0. This element is a_{1n}^n. Thus, in order to calculate the elements of the diagonal $\hat{a}_{i,i+k}$ of matrix $\hat{A}$, we only need elements of the powers of A that are less than or equal to k.

Next, we will see how many elemental operations, sequential and parallel operations, we need in order to calculate those elements.

Let d_k be the diagonal k for $k \in \{1,\dots,n-1\}$ of any matrix $A=(A_{ij})$. This diagonal has $n-k$ elements that are

$$d_k(A)=\{a_{j\,j+k}\}_{j=1}^{n-k}$$

The computational cost of computing $d_m(A^p)$, having computed before A^{p-1}, is $m-p+1$ sequential operations and $m-p$ parallel operations, where $p \le m$; in other cases the result is zero, meaning there is no computational cost. If the number of elements in such a diagonal is $n-m$, the computational cost to compute it is then $(m-p+1)(n-m)$ sequential operations and $(m-p)(n-m)$ parallel operations.

The number of sequential operations to compute $\hat{A}$ is

$$\sum_{m=2}^{n-1}\sum_{p=2}^{m}(m-p+1)(n-m)=\frac{(n+1)n(n-1)(n-2)}{24} \tag{1}$$

The number of parallel operations to compute the generalized powers of A is

$$\sum_{m=2}^{n-1}\sum_{p=2}^{m}(m-p)(n-m)=\frac{n(n-1)(n-2)(n-3)}{24} \tag{2}$$

In order to compute $d_m(\hat{A})$ we have to sum the non-null diagonals $d_m(A^p)$, i.e. $m-1$ diagonals. As these diagonals have $n-m$ elements, the number of parallel operations to sum the generalized powers of A is

$$\sum_{m=2}^{n-1}(m-1)(n-m)=\frac{n(n-1)(n-2)}{6} \tag{3}$$

Summing the results of 2 and 3 we obtain the total number of parallel operations that requires the calculation of $\hat{A}$.

$$\frac{n(n^3-2n^2-n+2)}{24}$$

The total number of elemental operations, counting parallel and sequential ones, is

$$\frac{n(n^3 - 2n^2 - n + 2)}{12}$$

As a remark we can say that this algorithm for calculating the trust values for n entities is of the order four times the order of the operators $\odot$ and $\oplus$. In the case they are both linear, the complexity is $\mathcal{O}(n^4)$.

3. PROCESS RATING COMPUTATION

Whenever we propose to rate processes, we suggest that the rating of sub-processes will be ideally assigned by the process that launches them. The fact is that we cannot enforce that this rating process is carried out by actual application, then in most occasions the system will have to assign them automatically based on some user parameters.

In order to make our framework work, this rating assignment must hold the following property: all ratings assigned by a given process must sum less than one.

We can see this rating as a quota of the resource that is granted to this sub-process, when it is launched by its parent. The quota is a real number in the interval [0,1]. A process expresses the quota assigned to its sub-process relatively to the actual rating it has been assigned.

In order to compute the quota of a certain process or sub-process, we will use the product of all the quota of the links in the chain from the root process to the sub-process we want to compute minus the quota that this process has assigned to its own sub-process. In fact, for each process we can distinguish the delegated quota, which is the quota that reaches this process through the execution path, and the actual quota of this process that is computed by subtracting the quota assigned to other process from the delegated quota of this process. Loops have no sense in our rating model because we are modelling execution paths.

All the quota assignments are managed in a central server that stores them as a matrix.

If we establish an equivalence between nodes or organizations and states, and between the quota that an entity X hands over to Z and the statistical concept of transition from one X to Z, our particular problem could be modelled as a discrete Markov's chain where the number of phases or stages corresponds to the length of the chain that we consider. Markov's chain has been used in different fields in security such as Trust Management [16], Privacy [17], Intrusion Detection Systems [18], etc. In this work we use them in order to reflect the assignment of ratings in a compatible way with the trust system that we developed.

3.1. Computational model

The initial organization or entity that first holds all the quota is called the *initiator* and it will be the initial state of the Markov's chain. The values of the quotas could be placed in a matrix such that the addition of the elements in each row is lower than or equal to one ('almost' a stochastic matrix). The reason is that the addition of all quota could never be greater than one, but it can be less, i.e. there could be states without any assigned quota. Therefore, in order to make this matrix a stochastic matrix we should add an additional state, namely, the state of the non-assigned quota.

This new state will mean a new column and row for the matrix of the states where the values in the columns are calculated in such a way that the addition of the values in the row is one. If we call this matrix A, the associated stochastic matrix A^* is as follows:

$$\left(\begin{array}{c|c} & 1-\sum_{j=1}^{n} a_{1,j} \\ A & \vdots \\ & 1-\sum_{j=1}^{n} a_{n,j} \\ \hline 0\ \ldots\ 0 & 1 \end{array}\right)$$

The state of ‘non-assigned’ quota is a non-transition state as once it has been reached no other state can be reached afterwards.

This matrix can be expressed in its canonical form as

$$\left(\begin{array}{c|c} A & R \\ \hline 0 & I \end{array}\right)$$

where A is the matrix of quota delegation that contains all the transition states; R is the matrix that contains the non-assigned quota by entities and I is the identity matrix of length one. As there are no loops in the quota delegation graph, the matrix A is upper triangular or at least we can make it upper triangular by reordering the nodes using a topological sort algorithm over the quota delegation graph. Therefore, the matrix of the chain, N, is calculated as follows:

$$N=(I-A)^{-1}$$

Next we will show that the element n_{ij} of N is the share of quota that entity i hands over to entity j.

An element $a_{ij}^{(k)}$ of matrix A^k represents the percentage of quota that node i hands over node j indirectly, if we only consider chains of length k. In addition, $A^n=0$ if n is greater than or equal to the size of the matrix, as this is an upper triangular matrix. Therefore, we can define matrix $\hat{A}$ as

$$\hat{A}=\sum_{i=1}^{\infty} A^i=\sum_{i=1}^{n-1} A^i$$

The elements of $\hat{A}$ can be calculated in the following manner:

$$\hat{a}_{ij}=\sum_{s=1}^{n-1} a_{ij}^{(s)}$$

$$\hat{A}=(\hat{a}_{ij})$$

The elements $\hat{a}_{ij}$ of $\hat{A}$ are the addition of all the quota that node i hands over to node j for chains of any length.

Next we will show that $N = I + \hat{A}$, i.e. $I + \hat{A}$ is the inverse of $I - A$. In order to do this, we will show that the matrix is invertible from the right-hand side (it is analogous from the left-hand side).

$$(I - A)(I + \hat{A}) = (I - A)\left(\sum_{s=0}^{n-1} A^s\right) = \sum_{s=0}^{n-1} A^s - \sum_{s=1}^{n} A^s = I + \sum_{s=1}^{n-1} A^s - \sum_{s=1}^{n-1} A^s - A^n = I$$

This gives us two ways of obtaining the percentage of quota handed over by entity i to entity j: either by calculating the element ij of matrix N or by calculating the element (i, j) of matrix $\hat{A}$.

In both cases, we can use the first row of those columns to form the vector of quota delegation from the initiator.

Definition 3.1 (*Quota Delegation Vector*). The Quota Delegation Vector is the vector v consisting of all the quota delegation values from the initiator to the rest of the entities. The first element is set to one.

In order to obtain the actual quota that remains in each entity, we have to subtract the quota it delegates from the quota it has been delegated. This can be easily done by multiplying the corresponding element of the quota delegation vector, v_i, by the element r_i of the column vector R.

By multiplying the column vector R by v, element by element, we obtain the quota distribution over the entities. The sum of all the elements of this vector is one.

3.2. Efficiency analysis of rating computation

The quota delegation model presents a feature that, in some cases, could be an advantage and, in some others, a disadvantage depending on the nature of the system. This feature is that if all the available quota has been assigned and we would be interested in assigning quota to a new entity, this should be taken away from the previously assigned quota.

Taking quota away could affect the entities that already had them assigned and therefore, the quota should be re-distributed again. This re-distribution will affect more the entities closer to the entity that is the root of the quota. Thus, if we establish an order where the entity origin of the quota is of order one and the other entities' order follows from the order how the quota is assigned, as higher the order, the less the impact the new distribution will have on this entity.

Taking into account all the above considerations, we can make some remarks concerning the complexity of the calculation of matrix $\hat{A}$. If we use the matrix N we can determine the complexity of the method by analysing the complexity of the calculation of the inverse of $I - A$.

This matrix is invertible and its inverse is $I + \hat{A}$. It is also upper triangular. In this case, we could solve n systems of simultaneous equations $(I - A)x^i = b_i$, where b_i are the consecutive columns of matrix I in order to calculate the inverse. This inverse will be the matrix, in columns, $(I - A)^{-1} = (x^{(1)} \| \dots \| x^{(n)})$. We can deduce that the inverse matrix is also upper triangular by observing the subsystem of the $n - i$ equations of each system,

$$\begin{pmatrix} 0 & 1 & \cdots & -a_{i+1\,n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_{i+1}^{(i)} \\ \vdots \\ x_n^{(i)} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

therefore, $x_j^{(i)}=0$ for $j>i$. Thus, resolving the remaining i equations of the system they can be simplified as follows:

$$\begin{pmatrix} 1 & -a_{12} & \cdots & -a_{1i} \\ 0 & 1 & \cdots & -a_{i+1\,i} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_i^{(i)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

This triangular system of i equations can be resolved by performing i^2 operations by using the substitution method. Thus, the final number of operations needed for calculating the inverse of $I-A$ will be

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

This means that by using the matrix N we can reduce the complexity to $\mathcal{O}(n^3)$.

In addition, since we calculate the columns of $(I-A)^{-1}$ independently, if there are changes in the assigned quota in node i, the columns with index less than i do not need to be re-calculated. Thus, the number of operations for updating the ratings can be computed as

$$\frac{n(n+1)(2n+1)-i(i+1)(2i+1)}{6}$$

4. MAKING ACCESS CONTROL DECISIONS

Access control decisions will be based, as mentioned above, on the rating of each process performing the request to the resource. The control mechanism is implemented as a lower bound for this rating. This lower bound may vary depending on the system overload or the resource availability.

All processes with a rating greater than or equal to the lower bound imposed by the access control policy must be served. In the case that two or more concurrent requests are performed, the rating will be used to order them accordingly. Processes with a higher rating will be served first.

4.1. Example

Let us assume three users in our system, Alice, Eve and Jack. Let us also assume that Jack is running two processes. Figure 1 shows a representation of the proposed scenario.

In this scenario, there are some direct trust relationships in the system with Alice and Eve. This trust relationship is measured with a level of $\frac{1}{3}$. They both trust in a third entity, Jack, with a value of $\frac{3}{4}$ each.

The trust matrix is then as follows:

$$A=\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{3}{4} \\ 0 & 0 & 0 & \frac{3}{4} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

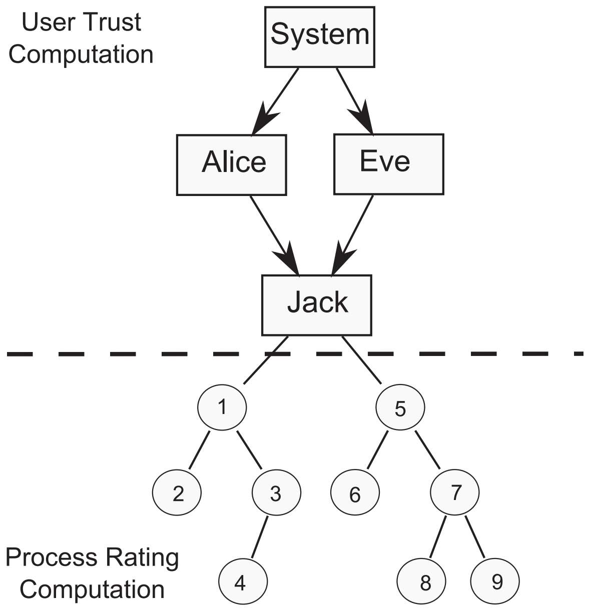

Figure 1. Sample.

If we use the product as sequential operator and the addition as parallel operator (see Section 2), the trust value associated with Jack will be $\frac{1}{2}$. This trust value will be the basis for rating the processes of Jack. Suppose that the system assigns to each process a rating according to the number of processes in the same level. In particular, the relative rating of each sub-process will be $\frac{1}{2}$ divided by the number of sub-processes. If we share the ratings in this way, $\frac{1}{2}$ will remain with the parent process. This means that this rating will be greater for the parent process than for the sub-processes.

The quota matrix is a follows:

$$B = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

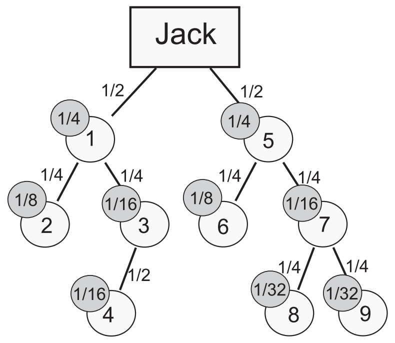

Figure 2. Distribution of ratings.

This matrix is strictly upper triangular because we ordered them in the same way as they were visited first by the Deep First Search algorithm. From this matrix we can obtain the stochastic matrix by including the non-assigned quota in the last row and column.

$$B^* = \left(\begin{array}{cccccccccc|c}
0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right)$$

The last row and column correspond to the state of non-assigned quota.

Let us assume that the ratings assigned to the scenario we described before are as shown in Figure 2.

The processes rating distribution is

$$1 \longrightarrow \frac{1}{4}$$

$$5 \longrightarrow \frac{1}{4}$$

$$2 \longrightarrow \frac{1}{8}$$

$$3 \longrightarrow \frac{1}{16}$$

$$4 \longrightarrow \frac{1}{16}$$

$$6 \longrightarrow \frac{1}{8}$$

$$7 \longrightarrow \frac{1}{16}$$

$$8 \longrightarrow \frac{1}{32}$$

$$9 \longrightarrow \frac{1}{32}$$

In case more than one of these processes performs a concurrent access request and given that these requests are granted, the priority for accessing is 1, 5, 2, 6, 3, 4, 7, 8, 9. If there are two processes with the same assigned ratings, we order them following the Deep First Search algorithm. Different strategies can be applied, the one we are using is a possibility.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a way of resolving the access control problem where several requests are performed concurrently. We took advantage of trust relationships among the users of the system and also process hierarchy.

We propose to enhance process hierarchy by assigning a trust rating to sub-processes. This rating can either be assigned automatically by the system or by the parent process itself.

Our model covers the priority queues' solutions and extends them in order to incorporate trust information.

Future work will consist of implementing this mechanism in the Linux operating system, by modifying the process scheduler.

We are also interested in investigating how the structure of users groups can be incorporated to our model together with the trust assigned value. These values could be assigned to the user himself or to the whole group.

As we are interested in applying this model to real applications, we would like to investigate other mechanisms for assigning ratings. These mechanisms can be based on actual parameters such as load of work, size of the process, waiting time, etc.

ACKNOWLEDGEMENTS

This work has been partially supported by the European Commission through the research project SPIKE (FP7-ICT-2007-1-217098), and the Spanish Ministry of Science and Education through the research project CRISIS (TIN2006-09242).

REFERENCES

1. Blaze M, Feigenbaum J, Lacy J. Decentralized trust management. *IEEE Symposium on Security and Privacy*, Berkeley, CA, U.S.A., 1996.

2. Blaze M, Feigenbaum J, Keromytis AD. *KeyNote*: *Trust Management for Public-key Infrastructures* (*Position Paper*). (*Lecture Notes in Computer Science*, vol. 1550). Springer: Berlin, 1999; 59–63.
3. Chu YH, Feigenbaum J, LaMacchia B, Resnick P, Strauss S. REFEREE: Trust management for Web applications. *Computer Networks and ISDN Systems* 1997; **29**:953–964.
4. Li N, Mitchell JC, Winsborough WH. Design of a role-based trust-management framework. *SP '02*: *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society: Washington, DC, U.S.A., 2002; 114.
5. Rao VN, Kumar V. Concurrent access of priority queues. *IEEE Transactions on Computers* 1988; **37**:1657–1665.
6. Burrows M, Abadi M, Needham RM. A logic of authentication. *ACM Transactions on Computer Systems* 1990; **8**(1):18–36.
7. Jøsang A, Ismail R. The beta reputation system. *15th Bled Electronic Commerce Conference e-Reality*: *Constructing the e-Economy*, Bled, Slovenia, June 2002.
8. Leiven R, Aiken A. Attack-resistant trust metrics for public key certification. *Proceedings of the Seventh USENIX Security Symposium*, San Antonio, TX, U.S.A., January 1998.
9. Jøsang A, Ismail R, Boyd C. A survey of trust and reputation systems for online service provision. *Decision Support Systems* 2007; **43**:618–644.
10. Available at: http://www.ebay.com [5 November 2008].
11. Agudo I, Fernandez-Gago C, Lopez J. A model for trust metrics analysis. *Fifth International Conference*, *TrustBus 2008* (*TrustBus'08*) (*Lecture Notes in Computer Science*, vol. 5185). Springer: Berlin, 2008; 28–37.
12. Available at: http://www.freehaven.net/ [5 November 2008].
13. Available at: http://www.openprivacy.org/ [5 November 2008].
14. Marsh S. Formalising trust as a computational concept. *PhD Thesis*, Department of Computer Science and Mathematics, University of Stirling, 1994.
15. Dijkstra EW. A note on two problems in connexion with graphs. *Numerische Mathematik* 1959; **1**:269–271.
16. Kamvar SD, Schlosser MT, Garcia-molina H. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th International World Wide Web Conference*. ACM Press: New York, 2003; 640–651.
17. Luh W, Kundur D. Distributed privacy for visual sensor networks via Markov shares. *Second IEEE Workshop on Dependability and Security in Sensor Networks and Systems*, Columbia, MD, 2006; 23–34.
18. Huaping R, Khanna L. Control theoretic approach to intrusion detection using a distributed hidden Markov model. *Wireless Communications* 2008; **15**(4):24–33.