An Authentication Protocol in Web-computing

Siman Wong

University of Massachusetts
Dept. of Mathematics and Statistics
Amherst, MA 01003-9315 USA
siman@math.umass.edu

Abstract

A web-computing system (WCS) *allows a host with limited resources to perform CPU intensive tasks by outsourcing the computations to external clients. But not every client is trusted, and redundancy in task assignment and auditing of results are needed to ensure the integrity of the results. This raises the question as to the efficiency and reliability of the system as measured against a given unit of the host's auditing time or cost. In this paper we propose a WCS with low overhead and has favorable error rate compared to a majority-voting scheme with similar efficiency. We can reduce the error rate by re-authenticating the results without having to resubmit any jobs, and we have an auditing strategy that in many cases is probabilistically better than random sampling.*

1. Introduction

The goal of a web-computing system (WCS) is to perform CPU intensive but parallelized tasks by out-sourcing the computations to external clients; the host's (limited) resources are devoted to the bookkeeping of outgoing tasks and the assembling of returning results. The popular SETI@home project [11] clearly demonstrates the potential of WCS, providing a processing rate of 60 tetraflops distributed over a million computers [1].

This new computing paradigm also brings with it a new challenge: how do we guarantee the integrity of the results if we do not know ahead of time the answers? Project managers for SETI@home have caught some of their volunteer participants *cheating*, i.e. claiming credits for tasks not completed, in order to move up their standing in the list of top CPU contributors ([1], [6]). The issue of cheating becomes even more critical if we want to deploy WCS commercially, where clients are paid a fee for the work they perform [6]. Another type of problem is *sabotage*, where fringe groups of individuals try to disrupt a WCS by submitting false results. Note that the issues we face here are different from those in the field of distributed fault tolerance with Byzantine faults (cf. [7]). To quote [9, p. 105], existing works on Byzantine faults tolerance focus mostly on *stopping faults*, where one or more of the processing elements or communication nodes simply stop generating or transmitting data, either temporarily or permanently; or on high performance parallel computation where the equipments are centrally located and the users are all trusted. Our focus, on the other hand, are on (again quoting [9, p. 105]) faults where the processors do not stop producing data, but instead producing bad data, possibly maliciously by hostile parties.

A common counter-measure for the problems we raised, as is implemented in the distributed computing managing software BOINC [2], is *redundancy* (or *majority-voting*) – assign the same task to multiple clients and take the most popular result as the correct answer. But this could be undermined by a group of *colluded*, untrusted clients. And since clients are often identified by their email or IP address, which are easy to fabricate, such a collusion attack could even be launched from a single source (called Sybil attack [3]). To foil such an attack we need to increase the degree of redundancy, which reduces the efficiency. Alternatively, or in addition, we can try to screen out the untrusted clients by *auditing* – assign trusted clients to redo a random sample of these jobs and compare the results with those turned in by the external clients – which increases the cost of the WCS.

A major source of the inefficiency of the majority-voting scheme is that it is highly *localized*: the effort expended on authenticating the result of one task has no bearing on any other task. We call this a *task-*

1-4244-0054-6/06/$20.00

centric model for WCS. We can visualize this by saying that the graphs for different tasks are not connected ($C_i(j)$ is the j-th client for task T_i):

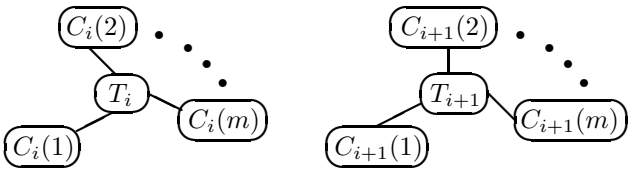

Figure 1. Majority voting scheme

We can augment the majority-voting scheme with *blacklisting*: the host keeps track of all results turned in by every client, and try to screen out the untrusted ones by random audits. But this would require multiple iterations of screenings before the portion of untrusted clients drop to an acceptable level. Moreover, the initial iterations would contain a higher portion of incorrect results, unless we discard all results turned in by untrusted clients discovered by audits, which reduces the efficiency of the system, or we backtrack and audit all these suspicious results, which could be costly. And because of the Sybil attack, even after we reject an untrusted client he can reenter the system under a new identity, thereby holding up the error rate of the system.

1.1 Related work

Sarmenta [9] studies WCS authentication as part of a *web-based volunteer computing* environment, and he gives theoretical analysis and experimental data for various WCS schemes, including redundancy-only and blacklisting. We will use these results and data as benchmarks for our work.

Several authors have approached guaranteed web-computing from the financial point of view. The payment scheme in [5] aims at discouraging dishonest clients by requiring each client to 'deposit' computations before payments, and by using a majority voting procedure that bans from the system every client from a group who turns in inconsistent results. The two payment schemes in [4] (see also [12]) embed special checkpoints within the tasks, and the client's payment depends on the extent to which these checkpoints are passed. However, a determined saboteur is not likely to be deterred by the prospect of financial losses; he could even turn the payment scheme in [5] around, submitting false results to drive out the honest clients (after fulfilling the initial 'deposit' to establish his standing in the system). Also, the schemes in [4] are not *transparent* – they require modification of the programs, and one of the lessons the SETI@home project taught us is that such modified codes are likely to be reverse-engineered [1].

The authors in [16] takes a game-theoretic point of view; their results suggest that unless we know ahead of time that few clients are untrusted and/or colluding, it could be just as cost effective to hire trusted, external clients to perform these tasks as it would to employ a distributed computing scheme with all the redundancy and auditing. On the other hand, their results also suggest that the cost of the host drops significantly if a single colluded group dominates a poll; this is compatible with the analysis of our WCS scheme.

The motivation behind the recent work of Szajda *et al* ([13], [14]) is the same as ours, namely to minimize the error rate for a given level of redundancy. These authors focus on one type of errors, namely collusion, and they offer two solutions. In [13] they propose to optimize collusion detection by using linear programming techniques to vary the degree of redundancy for different tasks. But this is done within the context of majority scheme, and for the collusion rates considered in [14, Fig. 1] or the detection rates considered in [14, Fig. 2], our protocol yields better performance. In [14] they propose an alternative task assignment scheme that resembles in some way our protocol: loosely speaking, clients are randomly assigned to vertices of a graph where any two clients are connected by an edge, and the host divides up several tasks into subtasks, each of which is identified with an edge of this graph. In particular, just like our protocol each subtask is assigned to exactly two clients. However, this scheme is applicable only to tasks that can be subdivided. And to achieve high collusion detection rate we need the group size (i.e. number of vertices) to be large, which significantly increases the bookkeeping overhead. Furthermore, if one pair of clients turn in incompatible results, this raises questions about the validity of the results of the rest of the clients in this large group. To resolve this issue we need to either (i) reject all results from clients who submits just *one* incompatible results, in which case the system is easily subject to attack by a saboteur; (ii) perform yet another vote (accept results from a client if say 80% of her results are compatible, in which case we are back to majority voting; or (iii) perform auditing, which is very expensive unless the host limits the number of audits, which minimizes its effectiveness, or keep the subtasks small, which leads to large group size and the aforementioned problems.

1.2 Contribution

In this paper we propose a new WCS with the following features:

- low overhead: the host does not keep track of past works, and each task is assigned to two clients only;
- transparency: no need to modify any tasks to run under our protocol;
- dynamic: clients can join or leave the system at any time.

We give a theoretical analysis of the efficiency and the error rate of our protocol, assuming maximal collusion (i.e. the worst-case scenario). Our protocol compares well against the basic redundancy-only scheme [9]. In addition, our protocol allows the host to estimate, *without* having to audit the results, the portion of untrusted clients and how often the untrusted ones actually turn in incorrect results; this information gives the host the option to *re-authenticate* the tasks at a different efficiency and error rate without having to resubmit any job. Finally, if the host chooses to audit the results, our protocol provides an optimal auditing strategy that yield in many cases a better fault-detection rate than random sampling.

We end this introduction by explaining the motivation behind our protocol, namely the concept of a *client-centric* WCS. Implicit in the majority-voting scheme is the assumption that those clients who are in the majority are trusted. In other words, the voting process for one task generates not just a consensus answer for the task but new information about the clients. However, we have no way of applying this piece of information to the voting for any other tasks. This suggests that we study WCS where the clients are connected to each other via *common* tasks. And to maximize the efficiency we need to minimize the number of clients assigned to a given task; equivalently, to minimize the valency of the graphs in Figure 1. The minimal valency of a connected graph is 2, so we are lead to consider WCS of the following form

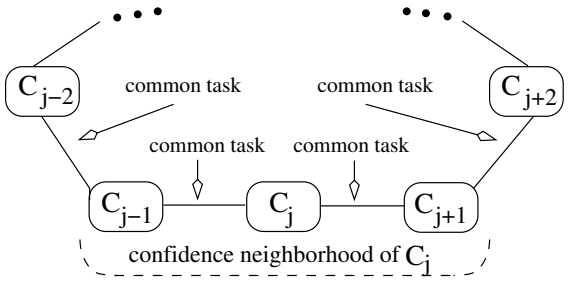

Figure 2. A client-centric WCS

Since each task is performed by only two clients, we can no longer rely on majority voting to generate a consensus result. Instead we *vote by reputation* – we declare that the results turned in by C_i are correct if *both* results of C_i agree with those of the neighbors of C_i. To increase the degree of confidence of our system we can enlarge these neighborhoods; for example, we can insist that not only should C_i agree with $C_{i\pm1}$, but that C_{i+1} should also agree with C_{i+2} as well. And unlike the majority-voting scheme, where the host must resubmit the task to additional clients if she is not confident of a particular vote, we can choose these confidence neighborhoods at the time of authentication, and we can re-authenticate the same collection of results at different level of confidence without having to resubmit any of these tasks. Finally, because each client is responsible for two tasks, we can audit one task and use the confidence neighborhood to *probabilistically audit* the second task.

To paraphrase the authors in [13], we do not claim that ours is the 'best' WCS protocol. With WCS packages such as BOINC, where authentication is implemented as a separate module, the host has the option of using different protocols with contrasting efficiency rate and error rate, so in practice the host can pick the protocol whose characteristics match her targeted performance and potential clients pool. Our protocol is an attractive option in an environment demanding low bookkeeping overhead, flexibility, and competitive performance using a clients pool that is not too untrusted (under $< 40\%$).

In Section 2 we state the precise hypotheses of our theoretical model. After describing our protocol in Section 3, in Section 4 we analyze the error rate and efficiency of our protocol, and we compare our protocol with the majority-voting scheme. Section 5 describes our probabilistic auditing scheme, and Section 6 concludes.

2 Model hypothesis

Denote by $\mathcal{T}$ a collection of tasks, and $\mathcal{C}$ a collection of clients. For the rest of this paper we make two hypotheses on $\mathcal{T}$:

(T1): one cannot correctly predict with high probability the output of a given task in $\mathcal{T}$ without actually performing the computations.

(T2): if a given task is assigned to two clients, it is computationally efficient for the host to decide if the two results are the same.

Numerical integration is a good example of tasks that satisfy (T1). *Database search* can be easily modified to fit hypothesis (T1) as well. For example, suppose we need to find a match within a given range of a database of names, indexed by ID numbers. We stipulate that the output of the search be made up of two components: the first component is either a 'yes' if we find the name (plus the record of this entry), or a 'no'; and the second component is the sum, modulo a given large integer (prescribed by the task), of the ASCII of every letter of every first name within the range. This output format serves to prevent *false-negative* results, and it can be trivially adopted for many similar tasks, such as inverting one-way functions by brute force search. Note also that the only restriction in our task assignment protocol is the condition (1); in particular, we can assign different types of tasks in our queue so long as they satisfy (T1).

Hypothesis (T2) is clearly satisfied by discrete, deterministic tasks such as the modified database search described above. The issue becomes more subtle, however, for numerical computations. A recent study [15] documents a real-life implementation of BOINC for protein-foldings involving simulations that are sensitive to initial conditions. The authors observe that the same Fortran code yields noticeably different results if they use the same hardware but different operating systems (Linux vs. Window), or the same operating system but different hardwares (Pentium 4 vs. AMD). Our protocol is not intended to address this issue, but we wish to bring this to the attention of the WCS community, and if necessary, follow the suggestion in [15] of distributing computations that are sensitive to initial conditions only among clients with identical hardware and software configurations.

We assume that the clients come in two types: the trusted ones and the untrusted ones. Furthermore, we assume that

(C1): the trusted clients always turn in correct results;

(C2): the untrusted clients are randomly distributed in the queue;

(C3): the probability that an untrusted client would turn in incorrect result for any given task is the same for all clients and for all tasks;

(C4): if an untrusted client is assigned multiple tasks, his decision to turn in incorrect to turn in incorrect result for each task is independent of each other;

(C5): the untrusted clients are *maximally colluded*: if two untrusted clients are assigned the same task and if they both decide to falsify a result, then they both turn in the *same* incorrect result.

In Section 4 we will analyze the performance of our protocol based on these hypotheses, noting that (C5) will always guarantee the worst case scenario.

3 Description of the Protocol

3.1 Task Assignment

We now construct an ordered task queue $\mathcal{Q} = \{Q_1, Q_2, \ldots\}$ as follows. To each slot $Q_i \in \mathcal{Q}$ we assign a random client $C_i \in \mathcal{C}$ and two distinct tasks $L_i, R_i \in \mathcal{T}$, called the *left task* and the *right task* for Q_i, such that

$$R_i = L_{i+1} \quad \text{for every } i \geq 2. \tag{1}$$

Note that $\mathcal{Q}$ is *dynamic*: we can add or reject clients as we go along. Also, the same client could be assigned to multiple queue slots, i.e. we allow $C_i = C_j$ for some $i \neq j$. Similarly, $\mathcal{T}$ is dynamic, and the same task could be assigned to multiple queue slots in addition to (1), i.e. we allow $L_i = L_j$ or $R_i = R_j$ for some $i \neq j$.

3.2 Authentication

To simplify the notation, we write $[T]$ for the result of a task $T \in \mathcal{T}$. There are three levels of authentication:

Level One. For every $i \geq 2$, check to see if

$$[R_{i-1}] = [L_i] \quad \text{and} \quad [R_i] = [L_{i+1}]. \tag{2}$$

If so then declare that *both* $[L_i]$ and $[R_i]$ are *correct to level one.*

Level Two. For every $i \geq 3$, check to see if (2) and at least *one of* the followings conditions holds:

$$[R_{i-2}] = [L_{i-1}]; \tag{3}$$

$$[R_{i+1}] = [L_{i+2}]. \tag{4}$$

If so then declare that *both* $[L_i]$ and $[R_i]$ are *correct to level two*.

Level Three. For every $i \geq 3$, check to see if (2) as well as *both* of (3) and (4) hold. If so then declare that *both* $[L_i]$ and $[R_i]$ are *correct to level three*.

By extension, we say that a client Q_i is *trusted to level* j if both $[L_i]$ and $[R_i]$ are correct to level j.

Remark 1. Note that the level one authentication procedure leaves unattended the queue slot Q_1. There are two ways to handle this:

(i) take C_1 to be a trusted client and accept as correct $[L_1]$ and $[R_1]$;

(ii) assign L_1 to be the right task at another queue slot, i.e. $R_j = L_1$ for some $j > 1$.

In case (i) we can use the correctness of $[R_1]$ to audit subsequent tasks. In case (ii) we effectively 'close up' the task queue $\mathcal{Q}$, turning it into an ordered *loop* and authenticate accordingly. Similarly, to authenticate L_2 and R_2 to level two, we can either assign both Q_1 and Q_2 to a trusted client, or close up $\mathcal{Q}$ so that $R_j = L_1$ and $R_{j-1} = L_j$; ditto for level three.

Remark 2. To authenticate Q_i we only need the results from its neighbors, not the entire queue. In particular, we do not have to wait for all the results to come in before commencing authentication.

Remark 3. If the same client appears more than once in the queue and his works are authenticated in one slot but not in others, we do *not* reject the ones deemed correct – the results deemed incorrect could be due to untrusted neighboring clients.

4 Performance analysis

4.1 Performance metrics

To analyze the performance of our protocol, we pick a random, finite, consecutive sequence $\mathcal{Q}'$ of tasks from $\mathcal{Q}$ and compare the number of results that are genuinely correct verse those that are deemed correct but in fact are not. The following quantities will be the bases of our performance metric:

- N: number of tasks in this random, consecutive sequence $\mathcal{Q}'$ of tasks;
- B: number of tasks in this random, consecutive sequence $\mathcal{Q}'$ of tasks;
- β: portion of untrusted clients $= B/N$;
- Cor: portion of Q_i both of whose tasks are performed correctly;
- $\mathrm{Cor_j}$: portion of Q_i that are declared trusted to level j;
- $\mathrm{False_j}$: portion of Q_i that are declared trusted to level j but submit at least one incorrect result;
- $\mathrm{Eff_j}$: efficiency of level j authentication = (number of Q_i trusted to level j)$/2N$ = $\mathrm{Cor_j}/2$;
- $\mathrm{Err_j}$: error rate of level j authentication = $\mathrm{False_j}/\mathrm{Cor_j}$.

The factor 2 in the definition of $\mathrm{Eff_j}$ is due to the fact that each task is assigned to two clients. Following hypothesis (C5), define

- α: probability that an untrusted client in $\mathcal{Q}'$ would turn in an incorrect result;
- φ: the *fault rate* of $\mathcal{Q}'$, i.e. the probability that an individual result is incorrect = $\alpha\beta$.

As in any WCS, the host does not know in advance which ones of the clients are untrusted. This is the reason why our definition of the efficiency rate Eff_j involves Cor_j, which can be measured by the host, and not Cor, which cannot. An interesting feature of our protocol is that from the authentication process alone (i.e. no auditing) we can determine α and β probabilistically; cf. Remark 4. As we will see later, this feature will allow us to optimize the performance and of our protocol as well as the (optional) auditing of the results.

4.2 Efficiency and error rate

Write the subqueue $\mathcal{Q}'$ as $\{Q_{i+1}, \ldots, Q_{i+N}\}$. We now work out the efficiency and error rate for level one authentication. The same method will yield the corresponding results for higher level authentications.

Theorem 1. *The efficiency and error rate for level one authentication are as follow:*

$$\mathrm{Eff}_1(\alpha,\beta) = \frac{1}{2}\begin{pmatrix} \alpha^4\beta^3 + (1-\alpha\beta)^2(1-\beta) \\ + (1-\alpha\beta)^2\beta(1-\alpha)^2 \\ + 2\beta^2\alpha^2(1-\alpha)(1-\alpha\beta) \end{pmatrix} + O\Big(\frac{1}{N}\Big)$$

$$\mathrm{Err}_1(\alpha,\beta) = \frac{\alpha^4\beta^3 + 2\beta^2\alpha^2(1-\alpha)(1-\alpha\beta)}{2\mathrm{Eff}_1(\alpha,\beta)} + O\Big(\frac{1}{N}\Big)$$

where the O-constants are absolute.

Proof. For $1 < j < N$, there are five scenarios under which Q_{i+j} will be declared correct to level one:
Case I. C_{i+j} is trusted.

Then the $C_{i+j\pm1}$ are either both trusted, or they are both untrusted and submit correct results. The

probability for this case is

$$\begin{aligned} & \Big(\frac{N-B-1}{N}+\frac{B}{N}(1-\alpha)\Big)\Big(\frac{N-B}{N}\Big) \\ & \times\Big(\frac{N-B-2}{N}+\frac{B-1}{N}(1-\alpha)\Big) \\ = \;& (1-\alpha\beta)^2(1-\beta)+O(1/N) \end{aligned}$$

with an absolute O-constant.

Case II. C_{i+j} is not trusted but both $[L_i]$ and $[R_i]$ are correct.

By hypothesis (C4), the correctness of each of the two tasks submitted by an untrusted client is independent of each other, so the probability that C_{i+j} satisfies the conditions in Case II is $\beta(1-\alpha)^2$. Also, both $C_{i+j\pm1}$ must be as in Case I. So the probability for Case II is

$$\begin{aligned} & (1-\beta+\beta(1-\alpha))^2\cdot\beta(1-\alpha)^2+O(1/N) \\ = \;& \beta(1-\alpha)^2(1-\alpha\beta)^2+O(1/N) \end{aligned}$$

with an absolute O-constant.

Case III. C_{i+j} is not trusted and $[L_i]$ is correct, but $[R_i]$ is not.

Then C_{i+j-1} is as in Case II, and (keeping in mind hypothesis (C5)) C_{i+j+1} must be untrusted and submit incorrect result. So the probability for this case is

$$\begin{aligned} & (1-\beta+\beta(1-\alpha))\beta(1-\alpha)\alpha\beta\alpha+O(1/N) \\ = \;& \beta^2\alpha^2(1-\alpha)(1-\alpha\beta)+O(1/N) \end{aligned}$$

with an absolute O-constant.

Case IV. C_{i+j} is not trusted and $[R_i]$ is correct, but $[L_i]$ is not.

We get the same probability as in Case III.

Case V. C_{i+j} is not trusted and both $[L_i]$ and $[R_i]$ are incorrect.

Then both $C_{i+j\pm1}$ are untrusted and both submit incorrect results. The probability is $\alpha^4\beta^3+O(1/N)$ with an absolute O-constant.

Note that each of these probabilities is independent of $1<j<N$, so for large N, up to an absolute $O(1)$-term the expected number of Q_{i+j} that are correct to level one can be computed by simply multiplying the corresponding probability by N. It follows that $\mathrm{Cor}_1(\alpha,\beta)$ is the sum of the five probabilities above. Case III, IV and V are the only scenarios under which Q_{i+j} would be declared correct to level one even though the results it submits are incorrect, so $\mathrm{False}_1(\alpha,\beta)$ is the sum of these three probabilities, and Theorem 1 follows. □

Remark 4. For a given implementation of our WCS protocol, $\mathrm{Cor}_j(\alpha,\beta)$ can be determined empirically from the authentication process alone with no auditing. For large values of N, these empirical values will match the main terms of the corresponding algebraic expression. We can then turn this around, substituting these empirical values into the polynomial expressions for $\mathrm{Cor}_1(\alpha,\beta)$ and $\mathrm{Cor}_2(\alpha,\beta)$ to solve for approximate values of α and β. More precisely, this will lead to finitely many choices for the pairs (α,β), and we can eliminate the extraneous ones by substituting each pair into the expression for $\mathrm{Cor}_3(\alpha,\beta)$.

Knowing the values of α and β will help improve auditing, if the host chooses to do so; cf. Section 5.

4.3 Majority voting

In this subsection we collect together several facts about a simple majority voting scheme. For more details see [9, Chap. 6].

Consider a majority voting scheme $\mathrm{Vote}_m(\alpha,\beta)$ where you assign the same task to $2m-1$ clients and accept as correct the common answer from at least m of them. If the fault rate is $\varphi=\alpha\beta$, then the error rate is [9, Eqn. (6.1)]

$$\mathrm{Err}_{\mathrm{Vote}}(\varphi,m)=\sum_{i=m}^{2m-1}\Big(\begin{array}{c}2m-1\\ i\end{array}\Big)\varphi^i(1-\varphi)^{2m-1-i}.$$

Since each task is performed by $2m-1$ clients, the efficiency rate is $\le 1/(2m-1)$. The next result tells us when we would hit this maximal rate.

Lemma 1. *The efficiency rate* $\mathrm{Eff}_{Vote}(\alpha,\beta,m)$ *of* $\mathrm{Vote}_m(\alpha,\beta)$ *is* $1/(2m-1)$ *when either* $\alpha\beta\le(m-1)/(2m-1)$ *or* $\alpha\beta\ge m/(2m-1)$.

Proof. Among the $2m-1$ random clients,

Prob(a given client is trusted) =
Prob(a given client is untrusted and turns in correct result) =
Prob(a given client is untrusted and turns in incorrect result) =

Note that these probability are independent of the clients, so the corresponding expected values can be computed by multiplying each probability value by $2m-1$. In the $\mathrm{Vote}_m(\alpha,\beta)$ scheme, we approve a result if we have at least m agreements; that means we need at least m correct or at least m compatible but incorrect results. Since the clients are assumed to be maximally colluded (recall hypothesis (C5) in §2) and we see that we need either $(2m-1)(1-\beta)+(2m-1)\beta(1-\alpha)\ge m$ or $(2m-1)\alpha\beta\ge m$, and the Lemma follows. □

4.4 Comparison with majority voting

Since every job is assigned to two different slots in the queue, the efficiency rate of our protocol will never

exceed 1/2. That would be the case for any WCS scheme that makes use of redundancy. But the for the same efficiency rate, our protocol compares favorably with the majority voting scheme Vote_m. To illustrate this we consider the case where Vote_m attains its maximum efficiency rate, namely $1/(2m-1)$.

Theorem 2. *Fix an integer $m \geq 2$, and consider those $0 \leq \alpha, \beta \leq 1$ so that $\text{Eff}_1(\alpha,\beta) = \text{Eff}_{\text{Vote}}(\alpha,\beta,m) = 1/(2m-1)$. Suppose N is large. Then $\text{Err}_1(\alpha,\beta) < \text{Err}_{\text{Vote}}(\alpha\beta,m)$ if $m > 4$, or if $2 \leq m \leq 4$ and β is less than the following bounds:*

m	2	3	4
minimal β	0.87267	0.39848	0.45030

Proof. Leaving out the $O(1/N)$-term, the equation $\text{Eff}_1(\alpha,\beta) = 1/(2m-1)$ becomes

$$\frac{1}{2}\left(\begin{array}{r}\alpha^4\beta^3 + (1-\alpha\beta)^2(1-\beta) \\ +(1-\alpha\beta)^2\beta(1-\alpha)^2 \\ +2\beta^2\alpha^2(1-\alpha)(1-\alpha\beta)\end{array}\right) = \frac{1}{2m-1}. \qquad (5)$$

Since $0 \leq \alpha, \beta \leq 1$, by calculus we find that the left side above takes its minimum value at $(\alpha,\beta) = (1,1/2)$ or $(1/2,1)$, with minimal value $1/8 > 1/(2\cdot 5-1)$. So $\text{Eff}_1(\alpha,\beta) = \text{Eff}_{\text{Vote}}(\alpha,\beta,m) = 1/(2m-1)$ only when $m \leq 4$, and of course we need $m \geq 2$ to have a vote. For any fixed m, the equation (5) defines a *curve* $C_{\text{Eff}}(m)$ on the (α,β)-plane, and the equation $\text{Err}_1(\alpha,\beta) = \text{Err}_{\text{Vote}}(\alpha\beta,m)$ defines another curve $C_{\text{Err}}(m)$. For each $2 \leq m \leq 4$, we plot these two curves on the same (α,β)-plane (see Figure 3 above) and we find that $C_{\text{Err}}(m)$ partitions the unit square into two regions inside each of which the difference $\text{Err}_{\text{Vote}}(\alpha\beta,m) - \text{Err}_1(\alpha,\beta)$ has constant sign (the darker curve is $C_{\text{Eff}}(m)$). Evaluate this difference at any interior point in these regions and we find that the sign is positive in the region to the right of $C_{\text{Err}}(m)$. The lower bounds for β in Theorem 2 now correspond, for $m = 2$, to the β-coordinate of the lowest point on the upper right hand portion of C_{Eff}; and for $m = 3$ and 4, the β-coordinate of the intersection point of these two curves (we readily check that the points so obtained for every m satisfy the conditions in Lemma 1). □

Remark 5. It seems unlikely that a WCS would be deployed in an environment where 39.8% of the clients are untrusted (especially with an efficiency rate of just $0.2 = 1/(2\cdot 3-1)$), so in practice our protocol offers better performance than Vote_m for comparable efficiency. And thanks to Remark 4, the host can estimate α,β without performing any audits. So if the host decides that $\text{Err}_1(\alpha,\beta)$ is too high, she can use a higher level authentication to reduce the error rate *without* having to resubmit any of the tasks. We are not aware of any other WCS scheme with a similar feature.

The table in Figure 4 compares the error rate for our protocol versus Vote_m for $m \leq 4$ for a sample set of α,β. Note that $\text{Err}_1(\alpha,\beta)$ for certain pairs of (α,β) in the table is higher than some of the corresponding $\text{Err}_{\text{Vote}_m}(\alpha\beta)$, but (as is predicted by Theorem 2) the corresponding $\text{Eff}_1(\alpha,\beta)$ is lower than that for Vote_m (recall that $\text{Eff}_{\text{Vote}_m} \leq 1/(2m-1)$).

5 Probabilistic Auditing

In the majority voting scheme, auditing the result of the client $C_i(j)$ (recall the notation in Figure 1) will determine whether this client is trusted in this instance; if she is not and if the host employs blacklisting, the auditing will then lead to the removal of $C_i(j)$ from the clients pool. To carry out this auditing the host needs to employ a trusted client to perform T_i; that means the host would have performed the task T_i without the input of any other clients $C_i(j')$ in the same voting block. We can summarize this by saying that it takes $2m-1 \geq 3$ clients, at least one of whom is trusted, to audit a single random client. Furthermore, as we pointed out in the introduction, the Sybil attack [3] neutralizes much of the benefits of blacklisting. Thus auditing Vote_m is an expensive process with limited benefits.

Turning now to our protocol, since each task is shared by exactly two clients, it only takes two clients, one of whom is trusted, to audit a single task. And since each client C_i is assigned two tasks, we can try to use the outcome of auditing (say) the left task L_i to predict the correctness of the right task R_i. This leads to a *probabilistic auditing* of our protocol. More precisely, we have *two* probabilistic auditing options, depending on whether the random client selected is declared correct or not to a given level. We now examine each of these two options. For simplicity we will focus on level one authentication.

Case: C_i is trusted to level one.

We would like to know the likelihood that the correctness of $[L_i]$ (as is confirmed by auditing) would imply the correctness of $[R_i]$. In other words, we need to determine

$$\frac{\text{Prob}\left(\begin{array}{l}C_i \text{ is trusted to level one and} \\ \text{both } [L_i],[R_i] \text{ are correct}\end{array}\right)}{\text{Prob}(C_i \text{ is trusted to level one})}.$$

In the proof of the Theorem 1 we saw that C_i is trusted to level one under exactly five scenarios. In particular,

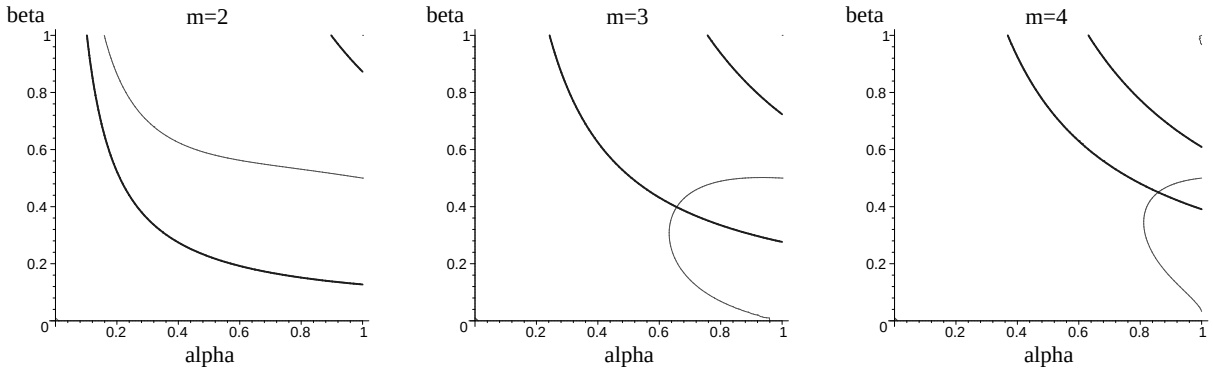

Figure 3. Plots of the curves $C_{\mathrm{Eff}}(m)$ and $C_{\mathrm{Err}}(m)$ for $m = 2, 3, 4$

β	α	$\mathrm{Err}_1(\alpha,\beta)$	$\mathrm{Eff}_1(\alpha,\beta)$	$m=2$	$m=3$	$m=4$
0.05	0.5	0.000674	0.457796	0.001843	0.000150	0.000012
	1.0	0.000145	0.428750	0.007249	0.001158	0.000193
0.10	0.5	0.002911	0.418625	0.007249	0.001158	0.000193
	1.0	0.001369	0.365000	0.027999	0.008559	0.002727
0.15	0.5	0.007079	0.382390	0.016031	0.003758	0.000920
	1.0	0.005465	0.308750	0.060749	0.026611	0.012103
0.20	0.5	0.013610	0.349000	0.027999	0.008559	0.002727
	1.0	0.015384	0.260000	0.103999	0.057919	0.033343
0.25	0.5	0.023006	0.318359	0.042968	0.016052	0.006238
	1.0	0.035714	0.218750	0.156249	0.103515	0.070556
0.30	0.5	0.035837	0.290375	0.060749	0.026611	0.012103
	1.0	0.072972	0.185000	0.215999	0.163079	0.126035
0.35	0.5	0.052736	0.264953	0.081156	0.040510	0.020949
	1.0	0.135039	0.158750	0.281749	0.235169	0.199845
0.40	0.5	0.074380	0.242000	0.103999	0.057919	0.033343
	1.0	0.228571	0.140000	0.351999	0.317439	0.289791
0.45	0.5	0.101457	0.221421	0.129093	0.078922	0.049760
	1.0	0.353883	0.128750	0.425249	0.406873	0.391712
0.50	0.5	0.134615	0.203125	0.156249	0.103515	0.070556
	1.0	0.499999	0.124999	0.499999	0.499999	0.499999

Figure 4. Error rate for Level One Authentication vs. Majority Voting

the denominator above is equal to the sum of the probability for these five cases, and the numerator, the sum of cases I and II. So up to an absolute $O(1/N)$-term, this ratio is

$$R_1(\alpha,\beta) := \frac{\left(\begin{array}{l}(1-\alpha\beta)^2(1-\beta)+\\(1-\alpha+\beta(1-\alpha))^2\beta(1-\alpha)^2\end{array}\right)}{2\mathrm{Eff}_1(\alpha,\beta)}.$$

In Figure 3 below we give the contour plot of $R_1(\alpha,\beta)$, with the top curve being $R_1(\alpha,\beta)=0.5$ and with the bottom one being $R_1(\alpha,\beta)=0.9$, in steps of 0.1. Using numerical integrations, we find that the area of the darkest region in Figure 3, viewed as a subset of the unit square, is 0.74950. To summarize, suppose N is large. Then for nearly 75% of the parameters (α,β), if auditing confirms one of the two tasks performed by a client who is trusted to level one, then with at least 90% certainty we can arrive at the *additional* conclusion that the other task is also correct. It is in this sense that this is a probabilistic auditing.

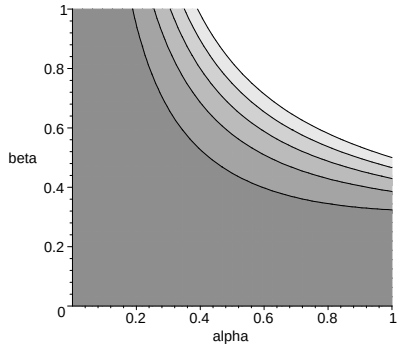

Figure 5. Plot of $R_1(\alpha,\beta)$

Case: C_i is not trusted to level one.

This time we need to determine

$$\frac{\mathrm{Prob}\left(\begin{array}{l}C_i \text{ is not trusted to level one,}\\ \text{but both } [L_i],[R_i] \text{ are correct}\end{array}\right)}{\mathrm{Prob}(C_i \text{ is not trusted to level one})}.$$

The same argument as in the proof of Theorem 1 shows that there are exactly five scenarios under which C_i is not trusted to level one. Moreover, in exactly two of these five scenarios both $[L_i]$ and $[R_i]$ are correct, namely when C_i is trusted, or when C_i is not trusted but submit correct results for both L_i and R_i. Up to an absolute $O(1/N)$-term we find that this ratio is

$$R_2(\alpha,\beta) := \frac{\beta^2(1-\beta)\alpha^2+\beta^3\alpha^2(1-\alpha)^2}{\left(\begin{array}{l}\beta^2(1-\beta)\alpha^2+\beta^3\alpha^2(1-\alpha)^2\\+2\beta^2\alpha^2(1-\alpha)(1-\alpha+\beta\alpha)\\+(1-\alpha+\beta\alpha)^2\beta\alpha^2\end{array}\right)}.$$

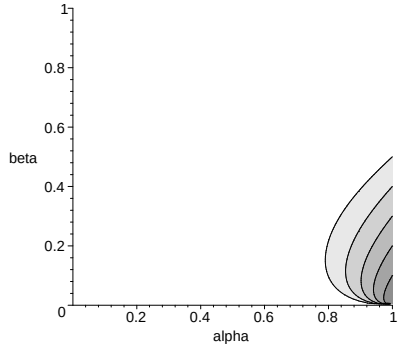

Figure 6. Plot of $R_2(\alpha,\beta)$

In Figure 4 we give the contour plot of $R_2(\alpha,\beta)$, starting with $R_2(\alpha,\beta)=0.5$ on the left of the picture and ending with $R_2(\alpha,\beta)=0.9$ on the right, in steps of 0.1. From the picture we see that we need α to be close to 1 in order to conduct a probabilistic auditing of clients that are untrusted to level one. Now, α measures how often an untrusted client submits an incorrect result. In light of hypothesis (C5), we can therefore interpret Figure 4 by saying that probabilistic auditing of untrusted clients is effective only when the untrusted clients are maximally untrusted. It is in this sense that this second type of auditing is compatible with the results in [16].

6 Conclusion and further work

We propose a WCS that is transparent, dynamic and with low overhead. It has a lower error rate than the majority-voting scheme with comparable efficiency, provided that at most 39.8% of the clients are untrusted. Our protocol also allows the host to estimate without auditing the proportion of untrusted clients, and how often these untrusted clients would submit incorrect results. These estimates give the host the option to trade off the error rate against the efficiency by re-authenticating the results without having to resubmit any jobs. And these estimates also allow the host to conduct probabilistic auditing.

The only storage requirement for the host are a list of the clients as well as the two tasks assigned to each client, and the authentication process only requires that the host check to see if the clients assigned the same tasks turn in the same result. It would be interesting to implement our protocol under the BOINC system [2] and compare the real-life performance of our protocol against the existing ones. It would also be interesting to combine our protocol with the linear programming strategy in [13] to further improve our

error rate, and to analyze further our probabilistic auditing strategy. For example, if two adjacent clients C_i and C_{i+1} are both trusted to level one, and if $[L_i]$ is found to be correct by auditing, it would be interesting to see how likely (and how often) can we deduce that $R_i = L_{i+1}$ and R_{i+1} are both performed correctly.

Acknowledgments

This research is supported in part by a grant from the National Security Agency. We would like to thank Professor Rosenberg for bringing to our attention the SETI@home problem, and for many useful discussions and constructive comments. We also would like to thank Professor Levine and Mr. Yurkewych for useful discussions.

References

[1] D. P. Anderson, Lesson learned from SETI@home. Presentation at *Collaborative Computing in Higher Education: Peer-to-Peer and Beyond*. Jan. 2002.

[2] D. P. Anderson, BOINC: a system for public-resource computing and storage. *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, 4-10. (2004)

[3] J. R. Douceur, The Sybil Attack. *Proc. 1st International Workshop on Peer-to-Peer System*. 2002.

[4] P. Golle and I. Mironov, Uncheatable distributed computations. *Topics in cryptology – CT-RSA 2001*.

[5] P. Golle and S. Stubbelbine, Secure distributed computing in a commercial environment. *Proc. financial cryptography 2001*.

[6] L. Kahney, Cheaters bow to peer pressure. *Wired News*, Feb. 15, 2001.

[7] Nancy A. Lynch, *Distributed algorithms*. Morgan Kaufmann Publishers, 1996.

[8] A. L. Rosenberg, Accountable web-computing. To appear in *IPDPS'02*.

[9] L. F. G. Sarmenta. *Volunteer Computing*. Ph.D. thesis, Dept. Elect. Eng. & Computer Science, MIT, March 2001.

[10] L. F. G. Sarmenta, Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems* **18** (2002) 561-572.

[11] SETI@home: Search for Extraterrestrial Intelligence at home. `http://setiathome.ssl.berkeley.edu`

[12] D. Szajda, B. Lawson, and J. Owen, Hardening functions for large-scale distributed computations. *Proc. 2003 IEEE Symposium on Security and Privacy*, 216-224. May 2003.

[13] D. Szajda, B. Lawson, and J. Owen, Toward an optimal redundancy strategy for distributed computations. *Proc. 2005 IEEE International Conf. on Cluster Computing (Cluster 2005)*.

[14] D. Szajda, J. Owen, B. Lawson, A. Charlesworth, and E. Kenney, An alternate multiplicity-2 task assignment scheme for distributed computations, in: *Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS 05), in conjunction with The 2005 International Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 05)*.

[15] M. Taufer, D. Anderson, P. Cicotti and C. Brooks, Homogeneous redundancy: a technique to ensure integrity of molecular simulation results. *14th Heterogeneous Computing Workshop (HCW 2005), in conjunction with IPDPS 2005*.

[16] M. Yurkewych, B. N. Levine, and A. L. Rosenberg, On the cost-ineffectiveness of redundancy in commercial P2P computing. ACM Conference on Computer and Communications Security (CCS), 2005.