Shadow Banning in Browser-based Volunteering Computing

Samuel Pélissier, Lucas Dupont, Dorian Lefeuvre, Nicolas Guillois

Shadow Banning in Browser-based Volunteering Computing

Samuel Pélissier, Lucas Dupont, Dorian Lefeuvre, Nicolas Guillois

Abstract—Browser-based volunteering computing projects are mainly used to perform scientific computations in heterogeneous clusters at a low cost. As for every community-driven approach, saboteurs can try to cheat the system for various reasons. In this paper, we propose to study whether such solutions could improve their performance and resilience by using shadow banning instead of a classic ban scheme. To do so, we have built a framework simulating a real system and studied the impact of shadow banning in relation with task types, saboteur rates, and detection techniques such as majority, m-first and credibility-based voting. Results show that shadow banning is overall more resilient, reducing the number of errors of detection by more than 33.5% in average. It also improves the server-side performance in a significant manner for saboteur rates between 0 and 20%.

Index Terms—Volunteering computing, Security, Web browsers, Distributed systems.

1 Introduction

Although supercomputers are used in science projects, they require a lot of money to build, maintain and operate. To counter these limitations and offer more open solutions, alternatives such as volunteer computing have been created.

As a part of a broader domain called citizen science [1], these solutions use the machines of contributors to perform distributed computations provided by scientists. The most famous implementation of such system is BOINC [2], where volunteers download a software and let it run in the background.

Coincidentally, the rise of browsers as a core component of every personal computer offers new and exciting possibilities for volunteer computing. Opening up to the browser world means accessing a huge number of devices such as mobile phones and tablets, without requiring any other software installation. Although their computing power is not comparable to specialised computers, their cheer number can outweigh the local lack of performance.

First using JavaScript [3] [4] [5] [6] [7], browser-based volunteering computing (BBVC) now takes advantage of new technologies such as the WebWorkers API[1] to create multiple threads. This helps unclog the main thread responsible of the page display and reduces the visible impacts of BBVC on visitors [8]. Another important breakthrough is the possibility to run native code, such as C and C++, directly in the browser. After various attempts at creating a durable solution [9] [10], Web Assembly was created in 2017 [11] and is since then widely supported. Although the performance improvements of such technology can be nuanced [12], it is still significantly better than JavaScript [13].

By essence, BBVC implies to run computation on a stranger's machine. This has two main consequences security-wise: the worker needs to trust the code running on his computer and the system can not trust the worker.

Although a saboteur can operate on his own, collaborative attacks are also worth considering [8]. Likewise, sybil attacks [14] are possible, given enough resources and depending on the detection techniques used by the BBVC system. For example, forging a new identity when the only verification is a cookie is way easier than validating a complex captcha [15]. However, BBVC systems usually aims at a minimal entry difficulty for workers to join, excluding complex identity management systems [16] [8]. No matter the restrictions, a system should be resilient not to a unique saboteur, but to a maximal percentage of them.

The simplest methods to discard incorrect results are based on voting. After a task has been done multiple times by different workers, the correct result is chosen with a vote. The most famous voting variants are m-first voting [2] and majority voting [17]. Although voting is easy to implement, it induces a significant overhead [18], [19], [20].

To improve the performance of the detection system, a new method named spot-checking has been proposed by Sarmenta *et al* [19]. The server randomly sends a task whose result is already known. If the worker returns a different value than expected, it is identified as malicious. All its previous results must be discarded and redone; this action is called backtracking. Optionally, a saboteur can be banned to ensure the next computation round will not contain as much errors. Multiple projects use and improve this idea [20] [21] [22].

In [19], spot-checking is actually not used alone but as part of a broader technique called credibility-based fault tolerance. This solution combines the benefits of voting and spot-checking to mathematically guarantee the results' correctness. Results are accepted only when their probability of being correct is above a threshold ϑ defined as $1 - \varepsilon_{acc}$ with ε_{acc} being an acceptable error rate.

In all these techniques, banning the saboteur and then backtracking the potential erroneous results is necessary. An

• *All authors are with Université de Bretagne Sud, more specifically studying at the École Nationale Supérieure d'Ingénieurs de Bretagne Sud.*

1. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

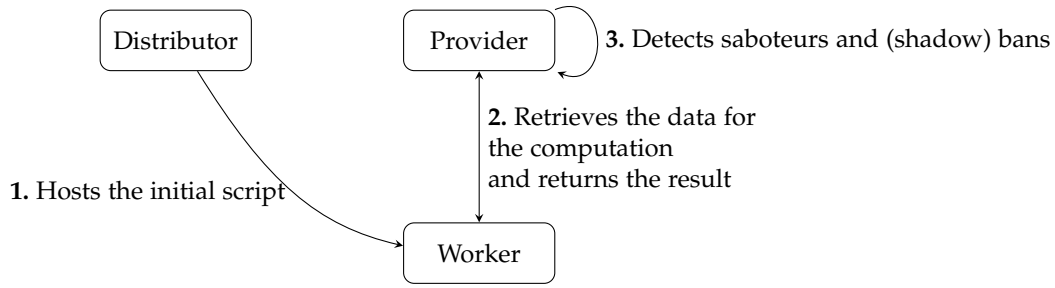

Fig. 1. High-level overview of a BBVC system

option not explored by the literature is based on shadow banning[2]. This process is the act of banning an user without them being aware of it [23]. A banned saboteur then interacts with the system exactly the same as an honest worker. This transparency reduces the incentive for a saboteur to change their identity, as they would do if banned from the system. Such control of the behavior of a saboteurs is useful to contain spammers, but could also improve the performance of BBVC systems.

We propose to study whether such solutions could improve their performance and resilience by using shadow banning instead of a classic ban scheme. To do so, we build a framework simulating a real system and study the impact of shadow banning in relation with task types, saboteur rates, and detection techniques such as majority, m-first and credibility-based voting.

One could argue this issue is also relevant to classic, BOINC-based projects. Although this is true, we believe BBVC has a bright future ahead of itself while facing some key specific challenges. For example, even if there are more available volunteers, their volatility is significantly higher. Moreover, the dwell time of website visitors follows a Weibull distribution which greatly impacts the system as a whole [24]. Thus, we want to specifically address the performance impacts of shadow ban in such settings.

2 Proposed approach

There are various options when creating a BBVC system; here, we provide an high-level overview of our approach. We use a client-server architecture with three main elements as seen in figure 1: a provider, at least one worker and a distributor to connect them.

2.1 Provider

The provider assumes multiple functions in a centralized manner. First, it stores the tasks, consisting of both code and data to be processed. When a worker asks for a new one, the provider uses a distribution algorithm to assign it to them, for example a First In First Out queue. Then, as computations results are received, a detection technique is applied, for example majority voting. At the end of this process, the provider knows if a worker is honest or a saboteur so it can start to ban if necessary. As seen in section 1, it implies backtracking the previous operations and actually (shadow) banning the saboteur.

All these actions and the relevant data are saved inside the provider's database for later use, for example to keep track of shadow banned users. Likewise, a centralized interface allows us to easily monitor the system's performance.

2.2 Worker

Honest workers and saboteurs interact with a BBVC system quite similarly. Both retrieve tasks and return either correct or falsified results to the provider. More specifically, coordinated and sybil attacks are needed to study the impact of shadow banning on detection techniques. A simpler approach is to use uncoordinated saboteurs, but it reduces the resilience of the system and is too far away from reality. Thus, we approximate their behavior by assuming they:

- Coordinate their attacks;
- Try to cheat every single time;
- Automatically join the system again after a ban;
- Only join again if banned; saboteurs do not reset by themselves.

2.3 Distributor

Finally, to connect workers and the provider together, a third-party can act as a distributor. In a BBVC system, the tasks are usually hosted by the provider, and the distributor only holds a minimal JavaScript snippet necessary to retrieve them. To ensure a large number of workers, this snippet is typically hosted by high-traffic websites.

The generic BBVC system presented previously allows us to monitor the server-side performance of shadow banning by studying the server and the provider's database. Likewise, we can determine which detection technique works best with shadow banning simply by changing a module in the provider and observing the variations.

3 Experimental setup

In this section, we present the experimental setup for the BBVC system described in section 2 and the various technical problems we try to solve. We refer to the result provided by a worker for a specific task as a "computation". For more low-level details and instructions to deploy our solution, please refer to the following URL: https://bob_project.gitlab.io/provider/.

2. Shadow banning is also known as "selective invisibility" and is linked to the usage of "twit bit" in online communities. See: https://ask.metafilter.com/117775/What-was-the-first-website-to-hide-trolls-activity-to-everyone-but-the-troll-himself

3.1 General settings

In the proposed implementation, both distributor and provider are hosted on the same physical server for the sake of simplicity. Similarly, our distribution algorithm is a Fist In First Out queue as it is the easiest to develop. More complex architecture and more fine-tuned distribution algorithms could be implemented but it should not influence the results in a significant way.

We write the code of distributed tasks in C, compile it to WebAssembly using the Emscripten toolchain[3] before sending it to the workers. C code converted to WebAssembly is fairly small and we launch it in a background thread thanks to WebWorkers. These measures only serve as performance improvements and, along using WebSocket for client-server status transfers, they can be viewed as following web development best practices. However, BBVC aims at exploiting the workers' machine without deteriorating the user experience [8] so any effort in this direction is worth taking.

3.2 Orchestration

Selenium[4] is used to automate visits on the distributor's website. This enables us to quickly start multiple workers on our machines. As the client-side implementation of a saboteur only requires to visit a specific URL, starting an honest worker or a saboteur is quite trivial.

Saboteurs are distributed on all the available machines used so they are not at an advantage or disadvantage by running on a specific computer. Thus, we recreate an heterogeneous cluster of a sufficient computing power. The various clients used are listed in table 1. Using all threads but one to keep an access on our systems, we reached 51 simultaneous threads or workers.

CPU	Number of threads
i7-7500U	4
i5-8600K	6
i7-6700HQ	8
i7-8550U	8
i7-8550U	8
i7-8565U	8
Ryzen 7 2700	16

TABLE 1
Computers used to conduct the experiments

A complete session of tests requires approximately 8 hours to complete. In order to smooth the experiments' results, we ran our tests 3 times. Although this is not enough to conduct a statistical analysis, it helps reducing the local spikes.

3.3 Metrics

There are multiple ways to judge the performance of shadow banning. In our case, a centralized BBVC system is used to reduce the infrastructure cost, so we monitor to which extend shadow banning reduces the server's load. As shadow banning can also be seen as a security measure, we further look at its impact on detection techniques. To do so, we collect various metrics; here, we only present the ones relevant to this paper. You can access all the data in this repository: https://gitlab.com/bob_project/results.

First, we study the network cost for a job, which is the sum of the data sent to workers, including the WebAssembly code, and the data retrieved. With the total number of computations sent to workers, it gives us an idea of the overall efficiency of the system in various settings. Each variation of configuration should influence these values, allowing us to compare a classic ban scheme to shadow banning. To further confront the two systems, we monitor the number of false attributions - either detecting an honest worker as a saboteur, or not detecting a saboteur. Finally, we collect the number of computations done after a shadow ban. All of these metrics help to study shadow banning regarding to the types of tasks and the detection systems.

3.4 Parameters

After describing which metrics to monitor, we need to define various parameters to test shadow banning in multiple settings. Then, we will be able to determine the optimal context to use it.

3.4.1 Type of detection

As the backtracking process changes based on the type of detection, we want to test a number of them. For our project, we selected three from the literature:

- Majority voting (5 replications);
- M-first voting (5 replications, first 3 results);
- Credibility-based voting and spot-checking, based on the works of Sarmenta *et al* [19].

More specifically, our BBVC system is considered without blacklisting as saboteurs can come back easily. Thus, the credibility of workers and their results follows the basic formula $Cr = 1 - \frac{k}{f}$ with k the number of previous spot-checks and f the fraction of saboteurs in the system. We use a threshold of 0.98 and a probability of spot-check of 0.25; these values were found empirically to be the most interesting for the dwell times of our project.

To determine the correct answer for a task, we compute hashes identifying each computation results provided by workers. These hashes are then used in one of the detection system described above. Workers who gave a wrong result are identified as saboteurs and are (shadow) banned; all of their work must be backtracked. Although backtracking can be adjusted to suit a specific type of detection, we opted for the simplest solution. Each time a saboteur is detected, all of the tasks they have been part of are reset and must be computed again.

3.4.2 Temporal distribution of workers

In a BBVC system, workers are highly volatile. Computations sometimes can not end and it is hazardous to assume a specific volunteer will stay connected. More specifically, the dwell time of workers follows a Weibull distribution. Based on [24], we create three categories of workers:

- Honest workers with a short dwell time (60 seconds);
- Honest workers with a long dwell time (300 seconds);

3. https://emscripten.org
4. https://www.selenium.dev/

- Saboteurs with a long dwell time (300 seconds).

To follow the Weibull distribution, 80% of workers belongs to the first category, while the others 20% are put in the second one. Moreover, each saboteur has a fixed pool of time; if they get banned before the timer ends, they automatically come back for the remaining visit duration. A smoother distribution could be used but the number of workers for our project is limited. More specifically, we use the data from table 2. Saboteur rates go from 0 to 50% but not over it, as majority voting is not usable passed this threshold.

Percentage of saboteurs	Number of saboteurs	Number of long workers	Number of short workers
0%	0	10	41
5.0%	2	10	39
10.0%	3	10	38
20.0%	6	9	36
30.0%	8	9	34
40.0%	12	8	31
50.0%	15	7	29

TABLE 2
Number of workers and saboteurs for each percentage of saboteurs

3.4.3 Type of tasks

Tasks are on a spectrum going from computing intensive to network intensive; some require more processing power while others need numerous round trips, and everything in between. In our experiment, we choose to test only two extreme scenarios to determine which setting fits a shadow banning scheme best.

To do so, we first implement a hash comparison algorithm using MD5 as a proof of concept. We send a hash and an interval of strings to workers, for example "aaaa" to "eeee". Then, they compute all the corresponding hashes and compare each one to the expected result. In this scenario, saboteurs only compute half of the interval of strings and return a falsified answer. By sending intervals of strings, we can fine-tune the required computation time. Doing so enables workers with a short dwell time to perform at least one task. For this purpose, we find that intervals of 10^7 hashes suit our system best.

For network intensive tasks, we change the color of a 500 pixels by 500 pixels PNG image using libpng[5]. Whereas data for hashes brute-force require around 30 bytes to transmit, each of the images weights between 200 and 500Kb. To visually demonstrate the good functioning of this proof of concept, honest workers return a blue image and saboteurs return a red one, each using the relevant RGB canal.

3.4.4 Saboteurs

As said in section 3.4.2, saboteurs stay on the page as long as the honest workers. However, their proportion varies from a test to another. Likewise, there can be a probability a saboteur comes back after they are banned. For our tests, we assumed a motivated saboteur would change their identity and request computations again 100% of the time.

5. www.libpng.org/pub/png/pngdocs.html

4 Results

In this section, we present our results, starting with the server-side performance implications of shadow banning, before studying its impact on detection techniques and finishing with the relevancy of the type of tasks.

4.1 Server-side performance

From the server's point of view, shadow banning works best with a small number of saboteurs. A saboteur rate of less than 30% provides a significant advantage over a classic system, before being outclassed in the following rates.

More specifically, we present in figure 2 the evolution of the network load for each saboteur rate during all the tests, sorted by types of bans. For example, for a 10% rate, around 140×10^6 bytes (140Mb) are transferred with shadow banning versus 131×10^6 bytes (131Mb) in a classic banning scheme; this roughly represents a 6.4% increase. This tendency is reversed in higher rates, as classic banning requires 13×10^6 bytes less (13Mb) for a 40% rate, meaning a 10% decrease compared to shadow banning. This behavior is directly linked to the number of computations sent to workers, as seen in figure 3. In average, shadow banning distributes around 14 computations less than classic banning for a 10% rate and 19 more for a 40% rate.

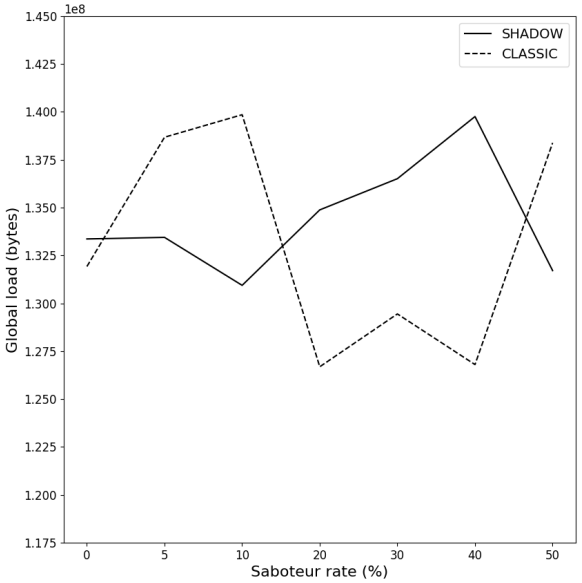

Fig. 2. Evolution of the network load for each saboteur rate

This can be explained by the quickness of distributing tasks, as shadow banned saboteurs do not have to re-join the computations after they are detected. Indeed, banned users in a classic system need to spot the change, and then log back in under a new identity. In our system, closing then loading a new browser window takes enough time to be noticeable and to induce these disparities.

This is particularly important, as the more time-consuming it is to create a new identity, the greater the

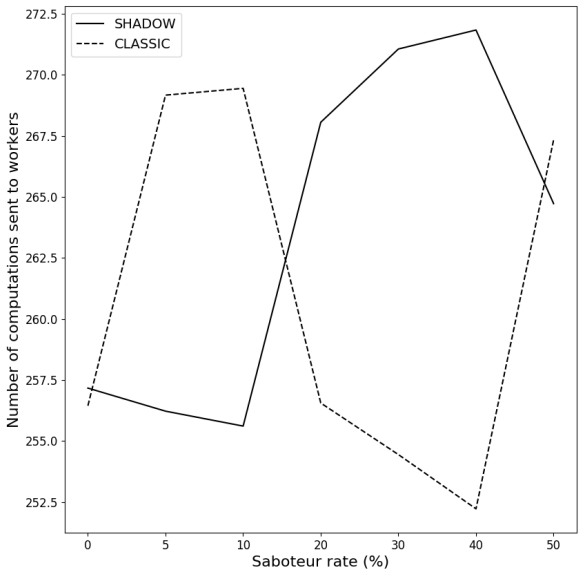

Fig. 3. Evolution of the number of computations sent to workers for each saboteur rate

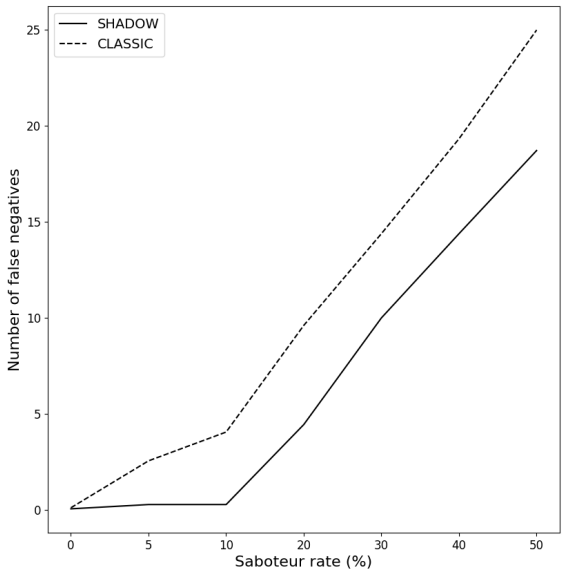

Fig. 4. Number of false negatives based on the saboteur rate, for classic and shadow banning systems

difference of performance between classic and shadow banning will be. Although our solution requires only half a dozen of seconds before reloading a saboteur, the period of time could be greater in more complex systems requiring to register a new account after a ban. However, a BBVC system aims to be as open as possible to take advantage of its numerous users. Using a complex and time-consuming process to enroll might discourage some potential new volunteers.

4.2 Synergy with detection techniques

Another interesting advantage of shadow banning is the smaller number of false attributions when detecting saboteurs, no matter the detection technique used. Figure 4 compares the number of times the provider falsely labels a saboteur as an honest worker (false negative) for a classic ban scheme and shadow banning. For example, shadow banning generates around 2 false negatives less than classic banning out of 2.5 for a 10% rate, and roughly 5 less out of 19.3 for a 40%. Studying the number of times the providers falsely labels an honest worker as a saboteur (false positive) in figure 5, we notice both systems are closer from each other in lower rates, but shadow banning is still more advantageous. Notably, for a 20% saboteur rate, only 7 false positives are raised using shadow banning versus 12.4 with a classic ban scheme. Overall, there are in average 35.83% less false negatives and 33.50% less false positives in a shadow banning system.

As shadow banned saboteurs do not try to rejoin the system under a new identity, there are less instances where they can be wrongly labelled. This is directly linked to the number of backtracks necessary: a classic system requires in average 8.7 operations more than shadow banning. Although shadow banning does not imply a faster detection

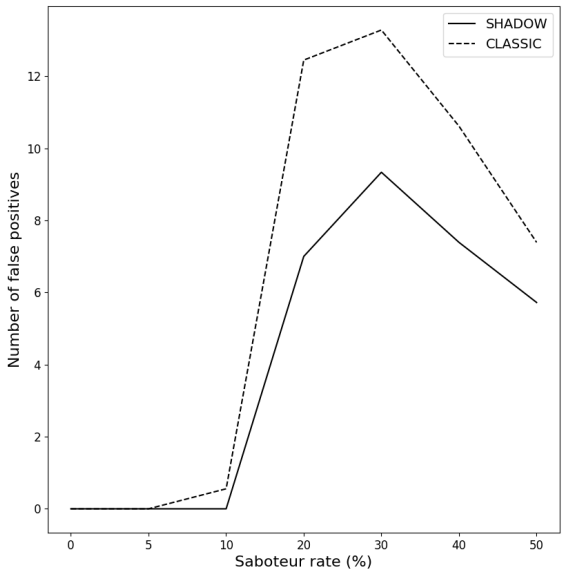

Fig. 5. Number of false positives based on the saboteur rate, for classic and shadow banning systems

and thus less backtracks directly, it naturally helps as saboteurs stay connected as long as long as possible. Hence, the backtracking operations required are done once, after the first and only detection.

Then, we study the number of false positives and false negatives in the context of various detection techniques. To do so, we subtract the number of false attributions with shadow banning to the one in a classic ban scheme for

each method: majority, m-first and credibility-based voting. For example, a difference of 10 means a classic ban scheme produces 10 more false attributions than shadow banning.

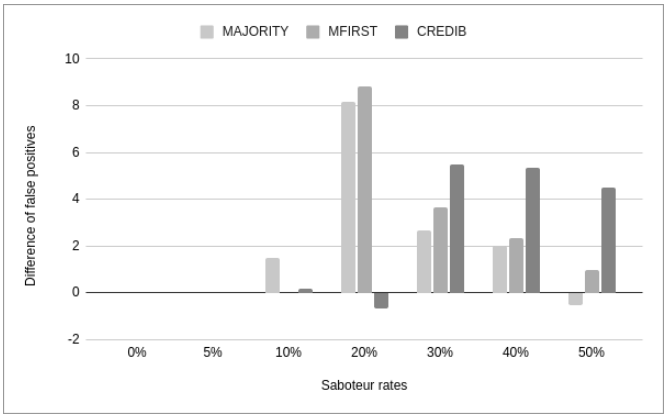

Fig. 6. Difference between the number of false positives in a classic and shadow banning system

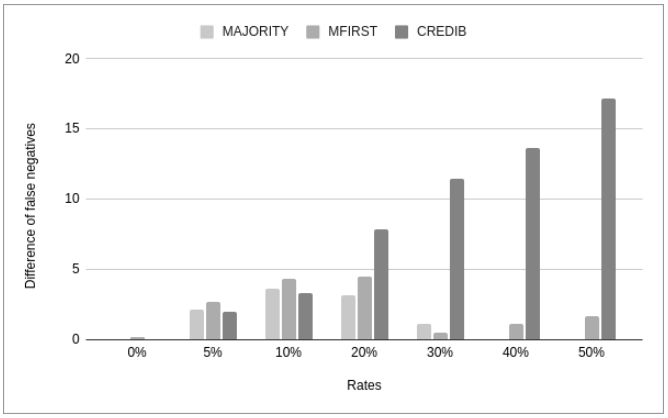

Fig. 7. Difference between the number of false negatives in a classic and shadow banning system

The varying factor shown in figure 6 is the distribution over various saboteur rates. For a 20% rate, majority and m-first voting are close to a 8 false positives difference out of 17.3 false positives for classic banning. However, it greatly decreases for other rates. In comparison, credibility-based detection is much more efficient starting from 30% saboteur rate; there are around between 4 and 6 less false positives using shadow banning with this technique. This phenomenon is also prevalent for false negatives as seen in figure 7. For higher saboteur rates, credibility-based voting is clearly more efficient, with around 14 less false negatives for 40%, whereas majority and m-first are close to their classic banning counterparts.

As the detection in majority and m-first voting techniques is based on the community of workers, a higher number of saboteurs reduces their efficiency. On the contrary, spot-check tasks used for the credibility-based system only depend on the studied worker. The number of false attributions from 30% to 50% saboteur rate with a credibility-based technique thus decreases compared to other detection mechanisms. In other words, shadow banning crystallizes the resilience of a detection technique.

4.3 Network or computing intensive tasks

Finally, we compare the two types of tasks used. Collected data does not seem to show a direct link between their nature - network or computing intensive - and shadow banning. However, time plays an important role. The color transformation of PNG images takes way less time than hash brute-forcing. This led PNG tasks to complete much faster, increasing the number of votes, detections and thus computations done after a shadow ban, as seen in figure 8. For example, for a 5% saboteur rate, there are in average 3.2 hash computations and 15.7 PNG computations done after shadow bans. This gap greatly increases in higher rates; for 40%, there are roughly 9 hash computations and 52.4 PNG computations. In average, 31 more computations were done after a shadow ban for PNG compared to hash brute-forcing.

4.4 Discussion

In light of the previous results related to false attributions, we can summarize the impact of shadow banning alongside the various detection techniques. As stated in table 3, this banning scheme suits majority and m-first voting best with saboteur rates below 30%. For higher rates, credibility-based detection proves to be more efficient. Thus, if the number of false attributions is an important criteria, one could adapt the detection mechanism based on the estimated saboteur rate.

Lastly, we compare the overall server-side performance of classic and shadow banning in table 4. Although the number of false attributions is always in favor of shadow banning, server-side performance heavily depends on the saboteur rate. In lower rates, shadow banning is more advantageous but this phenomenon is reversed after 30%. Thus, if reducing the server load is imperative, one may want to keep using a classic ban scheme in a highly unsafe environment.

In table 3, a plus sign means the studied context benefits from shadow banning whereas a minus sign means a classic ban scheme is more interesting. Using multiple plus signs is simply a way of comparing two or more criterion between themselves. In table 4, a plus sign indicates which banning scheme provides the best performance between the two.

Detection technique \ Saboteur rate	[0:30[	[30:50]
Majority	++	+
M-first	++	+
Credibility	-	++

TABLE 3
Comparing the efficiency of detection techniques while using shadow banning

Type of ban \ Saboteur rate	[0;30[	[30;50]
Classic banning	-	+
Shadow banning	+	-

TABLE 4
Comparing server-side performance for classic and shadow banning schemes

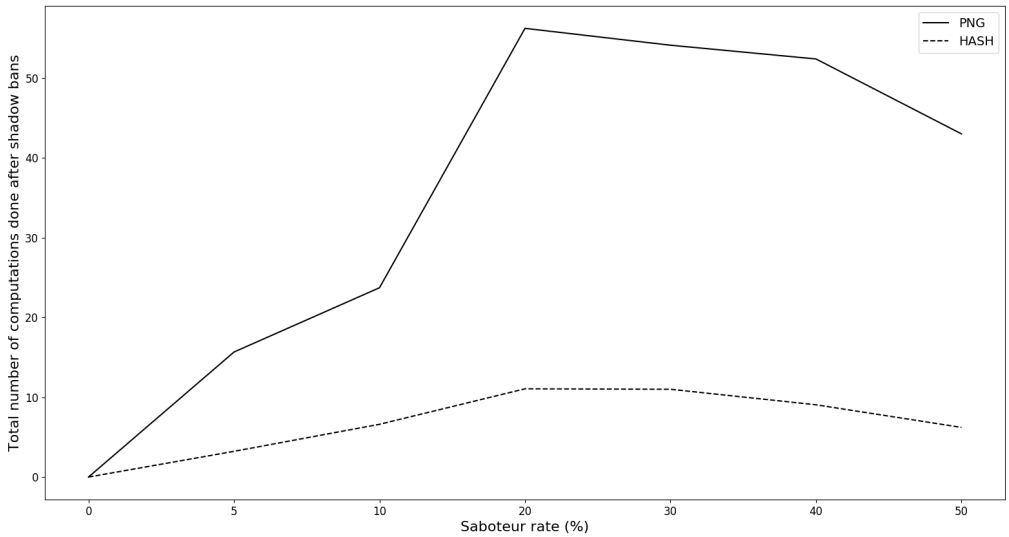

Fig. 8. Evolution of the number of computations done after a shadow ban, by type of tasks

5 Limits and Future Research

Although our model aims at recreating realistic settings, it is far from being an actual representation of a BBVC system as multiple parameters are approximated in order to conduct the experiments.

First of all, despite our best efforts to increase the number of workers, we could not come close to a realistic amount, compared to a slightly famous website. Finding a more organic but still relevant audience is a challenge for small-scale research projects and we settled for a repeatable approach. Duplicating each tasks 5 times over a 51 workers cluster reduces the precision of the results but is still close or higher than other works in the literature [25] [26].

Additionally, the dwell time of workers follows the Weibull distribution. This statistical analysis, while helpful to automate visitors' behavior, can vary from site to site. In our experiments, we make sure even short visits would be long enough to finish at least one computation. As even them can greatly impact the results, it would be necessary to adapt tasks to the actual minimal dwell time.

As stated in the section 2, we approximate the behavior of saboteurs. Allowing coordinated attacks is interesting to address as they do happen but it is quite overshadowed by saboteurs cheating every single time. Indeed, [27] hypothesizes that an attacker may want to alter a specific information, for example a single line of log containing incriminating data. In a less extreme scenario, we can imagine an attacker only sending false results from time to time. This behavior can be way more difficult to detect and to emulate without actual data to back the model up.

We also assume a saboteur automatically join the system again after a ban and do not reset their connection by themselves. This can also apply to honest workers, as they may re-join the system if they are wrongfully banned, or organically quit and come back later under a new identity. Based on our results, this is especially true, as honest workers are quite often excluded with higher saboteur rates. Again, a new analysis of field data may help the project calibrate more effectively their behavior.

Finally, we use our machines and our internet connections, which creates a somewhat heterogeneous cluster, but does not represent an actual model. A more realistic approach would be to follow field data for simulating the percentage of mobile phones, tablets and computers. Other than computing power, adapting the connection speed could be interesting. Either way, it impacts the workers' capacity to finish their computations in a timely fashion. Another approach is to adjust the minimal time required to the estimated power of the browser [8], also known as resource-aware scheduling [28].

In the future, studying the turning point around the 30% saboteur rate could be interesting. Although we have shown shadow banning can be advantageous both performance-wise and as a support to detection techniques, having a better understanding of when it lacks power could help improving it further. A more analytic study on the temporal nature of tasks, in opposition of our network/computing intensive classification, and their relation to shadow banning could also be conducted. As our project focused on browser-based computations, one could finally try shadow banning in more usual volunteering systems such as BOINC. The difference induced by the variation of dwell time in both settings should be investigated.

6 Conclusion

Shadow banning is overall more resilient than a classic ban scheme as it reduces the number of false attributions for all detection techniques studied. It also improves the server-side performance in a significant manner for saboteur rates between 0 and 20%. Considering these results, it can be an interesting alternative over conventional banning scheme in

BBCV systems promising a large number of visitors and thus low saboteur rates.

Acknowledgments

We thank our supervisor Dr. Vianney Lapôtre (Université Bretagne Sud), who provided a patient and thoughtful guidance for this first research project. We also thank Pierre-Marie Lechevalier for kindly proof-reading this document.

References

[1] J. Silvertown, "A new dawn for citizen science," vol. 24, no. 9, pp. 467–471, 2009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016953470900175X

[2] D. Anderson, "BOINC: A system for public-resource computing and storage," in *Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE, 2004, pp. 4–10. [Online]. Available: http://ieeexplore.ieee.org/document/1382809/

[3] F. Boldrin, C. Taddia, and G. Mazzini, "Distributed computing through web browser," in *2007 IEEE 66th Vehicular Technology Conference*, 2007, pp. 2020–2024, ISSN: 1090-3038.

[4] J. Klein and L. Spector, "Unwitting distributed genetic programming via asynchronous JavaScript and XML," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*. ACM Press, 2007, p. 1628. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1276958.1277282

[5] F. Konishi, M. Ishii, S. Ohki, R. Umetsu, and A. Konagaya, "RABC: A conceptual design of pervasive infrastructure for browser computing based on ajax technologies," 2007, pp. 661–672.

[6] T. Fabisiak and A. Danilecki, "Browser-based harnessing of voluntary computational power," vol. 42, no. 1, pp. 3–42, 2017, publisher: Sciendo Section: Foundations of Computing and Decision Sciences. [Online]. Available: https://content.sciendo.com/view/journals/fcds/42/1/article-p3.xml

[7] J. J. Merelo-Guervos, P. A. Castillo, J. Laredo, A. Mora Garcia, and A. Prieto, "Asynchronous distributed genetic algorithms with javascript and JSON," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 1372–1379. [Online]. Available: http://ieeexplore.ieee.org/document/4630973/

[8] Y. Pan, J. White, Y. Sun, and J. Gray, "Gray computing: A framework for computing with background JavaScript tasks," vol. 45, no. 2, pp. 171–193, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8105894/

[9] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native client: A sandbox for portable, untrusted x86 native code," in *2009 30th IEEE Symposium on Security and Privacy*, 2009, pp. 79–93, ISSN: 2375-1207.

[10] A. Zakai, "Emscripten: an LLVM-to-JavaScript compiler," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, ser. OOPSLA '11. Association for Computing Machinery, 2011, pp. 301–312. [Online]. Available: https://doi.org/10.1145/2048147.2048224

[11] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2017*. ACM Press, 2017, pp. 185–200. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3062341.3062363

[12] A. Jangda, B. Powers, E. D. Berger, and A. Guha, "Not so fast: analyzing the performance of webassembly vs. native code," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 107–120.

[13] S. Corts Sánchez, "Gocey - distributed evolutionary algorithms on ephemeral infrastructure."

[14] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[15] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Advances in Cryptology — EUROCRYPT 2003*, ser. Lecture Notes in Computer Science, E. Biham, Ed. Springer, 2003, pp. 294–311.

[16] D. Rotman, J. Preece, J. Hammock, K. Procita, D. Hansen, C. Parr, D. Lewis, and D. Jacobs, "Dynamic changes in motivation in collaborative citizen-science projects," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*. ACM Press, 2012, p. 217. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2145204.2145238

[17] S. Ryza and T. Wall, "MRJS: A JavaScript MapReduce framework for web browsers."

[18] T. Sakai and M. Fukushi, "A reliable volunteer computing system with credibility-based voting," vol. 24, no. 2, pp. 266–274, 2016. [Online]. Available: http://www.jstage.jst.go.jp/article/ipsjjip/24/2/24_266/_article

[19] L. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing systems," in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 337–346.

[20] K. Watanabe, M. Fukushi, and S. Horiguchi, "Optimal spot-checking for computation time minimization in volunteer computing," vol. 7, no. 4, pp. 575–600, 2009. [Online]. Available: http://link.springer.com/10.1007/s10723-009-9125-4

[21] G. Levitin, L. Xing, and Y. Dai, "Optimal spot-checking for collusion tolerance in computer grids," vol. 16, no. 2, pp. 301–312, 2019, conference Name: IEEE Transactions on Dependable and Secure Computing.

[22] N. Kopal, M. Wander, C. Konze, and H. Heck, "Adaptive cheat detection in decentralized volunteer computing with untrusted nodes," in *Distributed Applications and Interoperable Systems*, L. Y. Chen and H. P. Reiser, Eds. Springer International Publishing, 2017, vol. 10320, pp. 192–205, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-59665-5_14

[23] E. L. Merrer, B. Morgan, and G. Trédan, "Bug ou ban? une perspective topologique sur le shadow banning," p. 5, 2020.

[24] C. Liu, R. W. White, and S. Dumais, "Understanding web browsing behaviors through weibull analysis of dwell time," in *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10*. ACM Press, 2010, p. 379. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1835449.1835513

[25] R. Cushing, G. H. H. Putra, S. Koulouzis, A. Belloum, M. Bubak, and C. de Laat, "Distributed computing on an ensemble of browsers," vol. 17, no. 5, pp. 54–61, 2013, conference Name: IEEE Internet Computing.

[26] Y. Pan, J. White, Y. Sun, and J. Gray, "Gray computing: An analysis of computing with background JavaScript tasks," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 2015, pp. 167–177. [Online]. Available: http://ieeexplore.ieee.org/document/7194571/

[27] H. Ulusoy, M. Kantarcioglu, and E. Pattuk, "TrustMR: Computation integrity assurance system for MapReduce," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 441–450.

[28] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware adaptive scheduling for MapReduce clusters," in *Middleware 2011*, ser. Lecture Notes in Computer Science, F. Kon and A.-M. Kermarrec, Eds. Springer, 2011, pp. 187–207.