Risks in Anonymous Distributed Computing Systems

Michael J. Ciaraldi, David Finkel, and Craig E. Wills
Department of Computer Science
Worcester Polytechnic Institute
Worcester MA, USA 01609 USA
e-mail: ciaraldi@wpi.edu

Presented at the
International Network Conference 2000
Plymouth, UK
July, 2000

Abstract

Anonymous distributed systems consist of potentially millions of heterogeneous processing nodes connected by the global Internet. These nodes can be administered by thousands of organizations and individuals, with no direct knowledge of each other. Several approaches have been proposed to handle the technical aspects of such systems; this paper addresses some social, ethical, and legal aspects, particularly the potential risks, to nodes both within and outside such systems.

We describe the structure of anonymous distributed systems, then identify potential risks and where they occur within the structure. We then examine which risks can be addressed through existing techniques and technologies, and which require further study.

Keywords

Distributed computing, Risks, Security

1. Anonymous Distributed Computing Systems

A number of approaches have traditionally been used in research and practice to build distributed computing environments using a set of networked machines. These approaches include:

- Autonomous systems where machines run standalone, but users can explicitly access services on other machines, such as remote login or file transfer.
- Distributed operating systems which hide the details of the network and the existence of multiple machines from the user, providing the abstraction of a single virtual computer system. All of the machines in the system are under the control of a single administrative domain.
- Network file systems, the most common distributed computing environment, in which mostly autonomous machines share file systems located on remote file servers. Here, too, the machines are controlled by a single administrative domain.

Against the backdrop of these traditional approaches has arisen a new approach that seeks to solve distributed computing problems on a scale not possible with previous approaches. We refer to this approach as anonymous distributed computing (ADC) and these systems as anonymous distributed computing systems (ADCSs).

An ADCS consists of three types of nodes: distributor nodes for distributing pieces of a distributed computation, client nodes for executing these pieces and reporting results back to a distributor node, and portal nodes for serving as central sites where client nodes can be directed to distributor nodes. In general, these three types of nodes are not under the same administrative control.
ADCSs have several distinguishing characteristics:

- They consist of potentially millions of client nodes, each anonymously providing a piece of a distributed computation.
- The client nodes can vary widely in processing speed, memory capacity, and architecture.
- Each client node may be under the control of a different administrative domain.
- Client nodes may be unaware of each other.
- Client nodes may not always be available in the ADCS.
- Communication between client and distributor nodes is through the global Internet. This communication may be unreliable, intermittent, and at varying bandwidth.
- Client nodes may crash or unexpectedly withdraw from the ACDS at any time.
- A client node might participate in several ADCSs.
- Client nodes in an ADCS may participate voluntarily or they might receive payment, perhaps dependent on the quantity or quality of their computations.

Two general approaches are currently being used for anonymous distributed computing systems. In the first approach a client node first downloads an executable program from a portal node. When the client node wishes to actively participate in the distributed computation, it contacts a distributor node for specific data to use for processing and reports its results back to the distributor. At this point, the client node may request additional data from the distributor for execution of another computation piece. This approach is used by two ADCSs – The SETI@home project [10] and distributed.net [4]. The SETI@home project uses a distributor node to coordinate the work of client nodes to download radio telescope logs to search for evidence of extra-terrestrial intelligent life. Client nodes execute the program as a screen-saver so it is run when each node is otherwise idle. The distributed.net project is focused on solving DES encryption problems by using client nodes to test possible encryption keys. Client nodes execute the program as a low-priority background process. Owners of client nodes are not paid for participation, but part of a prize for solution of the distributed.net problem is given to the owner of the client node which solves the problem.

The second approach used by ADCSs is to execute Java applets on the client node. This approach is used by the POPCORN [9], Charlotte [1] and distriblets [2, 5] projects. In contrast to the first approach, these client nodes in these ADCSs do not download any executable code prior to active participation in the ADCS, but rather join an ADCS through a Web browser. Client nodes find distributors using portal nodes and then download a Java

applet along with data from the distributor at the time of participation. Client nodes execute the applet and report back results to the distributor, at which time they may request additional data for processing. These projects also do not include payment for work done by the client nodes, but literature on the POPCORN and distriblets projects [2, 5, 9] describes plans for micropayment mechanisms.

2. Types of Risks and Where They Occur

In this section, we identify a variety of risks that can occur in ADC. We note that the kinds of risks we are describing are well-known in the Internet community [6]. Our contribution is to examine how these risks affect anonymous distributed computations, and how the systems for ADC can be modified to avoid or ameliorate these risks.

Some of these risks can occur accidentally, by an unplanned malfunction of the network or one of the computers participating in the distributed computation, and some can occur deliberately, through the actions of a malicious user. In many cases, it is not necessary to distinguish between the accidental and deliberate cases, because the effect is the same.

One set of risks arises from the use of the Internet to communicate among the participants in the distributed computation. Data can be corrupted or lost during network transmission. Alternatively, since data on the Internet passes through computers under the administrative control of many different organizations, some organization or some individual with access at some organization may wish to delete or modify the data passing through their computers.

Another set of risks revolves around the anonymous nature of traffic on the Internet. Although each message on the Internet carries the address of the sender, it is relatively easy for senders to spoof their identity, so that each message they send does not carry the sender's correct address. Even if a message carries a true address, it may be difficult to know what organization controls that address. In the context of anonymous distributed computations, this identification problem means that neither the distributor of a computation nor the client in a computation can know with assurance with whom they are communicating.

Knowing the identity of the distributor can be crucial to assure clients that it is safe to install and run an application. Users might be willing to run a downloaded application on their computer because they have heard from others that they have downloaded and run the application without any harmful effects. They might have read about the application in a publication they consider reliable. But this assurance might be misplaced if they are mistaken about the identity of the distributor from which they are downloading the application. For example, the SETI@home computation discussed in Section 1 is distributed by the Web site http://setiathome.ssl.berkeley.edu/, and users connect to this site to download the SETI@home screensaver. A malicious individual could obtain a similar sounding URL and use that URL to distribute a harmful application

Knowing the identity of a client may be crucial as well. For example, a distributor may wish to limit the distribution of a computation to a restricted set of clients, either because the computation involves confidential data or because the distributors trust this restricted set of clients to execute the computation correctly. In such cases, it will be a problem for the distributor if it is not able to know the identity of the clients with which it is communicating.

Aside from the risks related to network transmission and distributor / client identification, there are additional risks related to the fact that in these systems the client and the distributor may be unknown to each other, and each may not know if the other is trustworthy.

As an example of a risk to the distributor, consider a system in which the distributor makes payments to clients for their participation in the distributed computation. The Popcorn system [9] includes an extensive system for payments to clients. The distriblets papers [2, 5] discuss the possibility of using a micropayment system for making small payments to clients. The risk to the distributor arises from the temptation for clients to provide spurious results in order to earn undeserved payments. For example, in the distriblet system, a client could replace the applet provided by the distributor with a client which returns a random number to the distributor instead of actually performing the required calculation.

A risk to the client arises if the distributor is untrustworthy. As discussed above, it's possible for the distributor to provide a program that will actually harm the client computer. But the distributor might provide a program that doesn't harm the client's computer, but performs a computation that the client considers unworthy. For example, the distributed computation might involve an illegal activity, like cracking an encryption scheme in order to commit an electronic theft. As another example, the distributor might provide a program that contacts a remote Web site repeatedly, taking part in a denial of service attack.

3. Dealing with Risks

3.1. Accidental Communication Problems

There is a chance that programs and data can be inadvertently corrupted *en route*. This can be easily handled by telecom protocols. For example, TCP includes packet checksums and automatic retransmission of corrupted packets. If this is not sufficient, additional checks, such as application-level error detection and correction codes, can easily be added.

Because Internet connectivity can be slow or intermittent, it is possible that a distributor will send out a request for computation and either never receive a response or receive it too late to be useful. Adding a timestamp to each request and response will enable the clients to decide whether or not to process the request, and the distributor to decide whether or not to accept the response. Some systems, such as Seti@Home, allow clients to participate if they know they will be disconnected from the Internet. In that case, the client retrieves and caches multiple requests, then saves the results and reports them later.

3.2. Deliberate Communication Problems

Deliberate interception and/or modification *en route* can be detected by encryption, e.g. by using IPSec [7]. But IPSec itself provides only a uniform protocol to support encryption and authentication over the Internet; administrative issues are equally important.

Modern encryption systems use two basic approaches: private key and public key methods.

- In a private key (symmetric) system, two parties that wish to communicate securely must share a secret key known only by the two of them.

- In contrast, in a public key (asymmetric) system each party has a matched pair of keys, public and private. Each party keeps its private key secret, but allows anyone to know the public key; this is safe because it is computationally infeasible to derive the private key from the public key. Someone wanting to send a secure message encrypts it using the receiver's public key; since this message can only be decrypted using the matching private key, only the intended recipient will be able to read it.

ADCSs cannot use a private key system; since the parties don't know each other, the distributor would have to generate a separate shared secret for every one of the millions of clients, and distribute them securely. The only practical solution is to use asymmetric (public key) encryption. Combining IPSec with the Internet Key Exchange (IKE) protocol [8] allows keys to be generated and distributed automatically and securely.

IPSec also includes the option for authentication, separate from whether the message is encrypted. The sender computes a hash function on the message, using a key. The receiver can be sure that only the true sender could have generated this hash.
The result is that, once IKE negotiation is complete, it is guaranteed that whatever machine is at the other end of the connection will continue to be at the other end—no other machine can impersonate it. But, at the start, how does each machine know the identity of the other machine?

Both encryption and authentication depend on reliably knowing the public key for the other party, that is, the party it is claiming to be. The current solution to the problem of confirming the public key uses a digital certificate issued by a certification authority (CA). But this has risks too:

- Can the CA be trusted? Perhaps it is run by a criminal organization or a hostile government. Even if the CA is legitimate, some of its employees could be corrupt.
- Can the CA really guarantee that this entity is who it claims to be? Will the CA require some form of identification, or perform some stronger check, before issuing a certificate? A law enforcement agency issuing certificates for its own staff would presumably know who they are and could vouch for their trustworthiness; a commercial CA might not be as careful about its clients.
- Even if the identity of the entity is known, is it someone who can be trusted to be either non-malicious or competent? And even if the organization is safe, does this apply to all its members? This could be especially important at a university, which has the same computers shared by many people with differing skills and agendas.
- Finally, certificates expire or need to be revoked (e.g. because the CA's security has been compromised), but this information takes time to propagate through the network.

3.3. Malicious Client Code

The possibility of a downloaded program damaging a client machine is minimized or eliminated when using Java applets, because the Java Virtual Machine executes applets in a "sandbox", an environment with limited privileges. For example, applets cannot read or write files on the client machine without explicit permission from the user. Note that the details of these protections depend on the version of Java, and are likely to change as the language

evolves [3]. In contrast, screen savers and ActiveX controls in Microsoft Windows do not have such protection. They execute with the same privileges as any other programs, which in practice means they have essentially unlimited access to all system resources.

Client programs running on Unix or similar systems would have whatever privileges the user who is running them possesses. Therefore, they should not be run by individual users. Rather, the programs should be run from a special account, analogous to the ones used by daemons, with only the privileges they need. Even when running with restricted privileges, a malicious client program can still incapacitate a computer or inconvenience its users. A Java applet could open multiple windows on the screen, send email with the client computer's return address, or consume so many system resources as to render the machine unusable.

A Java applet running in the sandbox has an additional limitation: it can only open a network connection back to the server from which it was loaded. This eliminates the risk of the applet being used to directly participate in a distributed attack (e.g. a denial-of-service attack from multiple locations), but greatly restricts the amount of parallelism the computations can achieve, because the distributor machine could act as a bottleneck.

Another approach would be to send the client code to clients in source form, so they could check it for maliciousness. Aside from the question of whether the clients have the time and expertise to do this checking, revealing the source code would make it almost impossible to guard against counterfeit client code.

3.4. Counterfeit Client Code

The Popcorn project discusses several approaches to overcoming the risk of having a client provide spurious answers. One approach is to send out each portion of the computation to several independent clients. If one of them is providing spurious answers, its results will not be the same as the results provided by the other clients. We note that this approach is expensive but generally applicable; although it is conceivable that several clients would conspire to defraud the distributor, the vast number of clients and the inability of a client to know which parts of the problem it will be assigned make this impractical. Another approach is to organize the computations so that it's easy for the distributor to check the answers after they have been returned by the clients. For example, if the distributed task were to factor a large integer into primes, it would be relatively easy to check whether the factorization returned as a result is correct or not (although not necessarily which factor is incorrect). This approach is less expensive than the preceding one, but it is not generally applicable to all computations. The question on both these approaches is whether the resources spent on checking exceed those gained by parallelism.

There are several approaches that could guard against a client substituting its own counterfeit client code, for whatever reason. The most promising is probably to have the distributor and client engage in some sort of challenge-response authentication scheme; only the true client code would be able to authenticate itself. It remains to be seen if this can be done. Since the object code of the client (whether in machine language or Java bytecodes) is visible, it could conceivably be decompiled and reverse engineered to form a convincing counterfeit. There is also the possibility that a third party could compromise the client machine, planting a "Trojan horse" which would later corrupt or replace the client code.

Any authentication scheme should include a *nonce* mechanism, ensuring that authentication of a particular piece of downloaded code expires in a short period of time. This reduces the window of opportunity for counterfeit code to successfully masquerade as the true client.

3.5. Portals

In several of the proposed anonymous distributed computation schemes, clients connect to a well-known central portal to participate in a computation. It would seem that such an approach would provide some additional protection to the client. However, the fact that the client is connecting through a well-known, and possibly trusted, portal doesn't necessarily mean that the distributor using that portal can be trusted. For example, in the current version of the distriblet system, anyone with a computation to perform may distribute it using the central server. It is possible to modify the system to require prior registration before a computation is distributed, but even if registration is required, that only guarantees that the individual distributing the computation can be identified, not that the computation is harmless.

What level of protection should be provided by using a central portal? There are two issues to consider: 1) what level of protection the client expects, and 2) what kinds of protection are technically feasible. With respect to point 1), the risk to users is certainly no greater than the risk of downloading software from a Web site dedicated to distributing shareware. Even if the administrators of the Web site examine the software they are offering for download, neither they nor the users can be certain that the software is completely harmless. With respect to point 2), the administrators of the portal site would find it difficult to examine a proposed distributed computation and know with certainty whether it is harmless, and even more difficult to determine whether its computation would be considered unethical by some users.

Even if the operators of a portal or distributor issue disclaimers, they must still exercise some sort of due diligence. Despite this, they may still be open to liability if their site has been used as an accomplice in nefarious activities.

4. Conclusion

Anonymous distributed computing systems share the risks of any distributed system, but present unique challenges. Some of these can be addressed with current technology, but many solutions await future research. The ultimate test of the seriousness of these risks is whether users are dissuaded from participating in anonymous distributed computations because of the risks.

5. Acknowledgements

We gratefully acknowledge the comments of Christof Parr and Richard Stanley from WPI, and of the INC2000 reviewers.

References

1. A. Baratloo, M. Karaul, Z. Kadem, and P. Wyckoff. "Charlotte: Metacomputing on the Web", *Proceedings of the International Conference on Parallel and Distributed Systems*, Dijon, France. September, 1996.
2. James F. Carlson, David V. Esposito, Nathaniel J. Springer, David Finkel and Craig E. Wills, "Applet-Based Distributed Computing on the Web," *Proceedings of the Workshop on Distributed Computing on the Web* (June, 1999), pp. 69 – 76.
3. Eva Chen, "Poison Java," IEEE Spectrum (August 1999). pp. 38 – 43.
4. Distributed.net, "Distributed.net Node Zero", http://www.distributed.net/, 1999.
5. David Finkel, Craig E. Wills, Kevin Amorin, Adam Covati, and Michael Lee, "An Applet-Based Approach to Large-Scale Distributed Computing", to appear in *Proceedings of the International Network Conference 2000* (July, 2000)
6. Warwick Ford and Michael S. Baum, *Secure Electronic Commerce*, Prentice-Hall PTR, Upper Saddle River, New Jersey, 1997.
7. Network Working Group, "Security Architecture for the Internet Protocol", http://www.ietf.org/rfc/rfc2401.txt, 1998.
8. Network Working Group, "The Internet Key Exchange (IKE)", http://www.ietf.org/rfc/rfc2409.txt, 1998.
9. "The POPCORN Project", http://www.cs.huji.ac.il/~popcorn/index.html, 1999.
10. "SETI@home: Search for Extraterrestrial Intelligence at Home", http://setiathome.ssl.berkeley.edu/, 1999.