Intrusion-Tolerant Parsimonious State Machine Replication *

HariGovind V. Ramasamy Adnan Agbaria William H. Sanders
University of Illinois, Urbana, IL 61801, USA
{ramasamy, adnan, whs}@crhc.uiuc.edu

Abstract

We describe a Byzantine-fault-tolerant state machine replication algorithm that reduces computation and communication costs in the fault-free case, and is reasonably efficient even in the presence of faults. Such an algorithm is practically significant, because failures are the exception than the norm, and much of a system's runtime is fault-free. The algorithm is geared towards applications that require Byzantine-fault tolerance, and also require that redundant processing and wasteful resource use should be reduced as much as possible (e.g., critical computations on the Grid).

1 Introduction

Intrusion tolerance recognizes the impracticability of defending a system against all attacks, and focuses on providing an acceptable QoS despite the compromise of some subsystems. A common way to construct an intrusion-tolerant system is to structure the system as a state machine, replicate the system on multiple nodes, and coordinate the nodes using a Byzantine-fault-tolerant replication protocol. Of course, this approach works only if the replica failure probabilities are not strongly correlated. Recent work in Byzantine-fault tolerance has focused on protocols that are practical and relevant to the implementation of real services (e.g., [1, 2, 5]). SINTRA [1] implements protocols based on threshold public-key cryptography for intrusion-tolerant replication over the Internet. PBFT [2] is a replication protocol that includes many optimizations, including the use of message authentication codes (MACs) instead of digital signatures. Yin et al. [5] observed that separating agreement on the order of request executions from the actual execution of requests can help improve confidentiality and also decrease the number of replicas needed to execute the requests from $3t + 1$ to $2t + 1$, where t is the maximum number of replicas that could be simultaneously faulty.

Like [1, 2, 5], our work also deals with Byzantine-fault-tolerant state machine replication that does not rely on synchrony assumptions for correctness. We explore another way to make Byzantine-fault tolerance useful for real services: reduction of computation and communication costs in the fault-free case. A Byzantine-fault-tolerant replication algorithm that reduces costs in the fault-free case would be practically significant, because experience shows that failures are the exception rather than the norm and that much of a system's runtime is fault-free. Our algorithm is geared towards applications that require Byzantine-fault tolerance, and also require that redundant processing and wasteful resource use should be minimized as much as possible. For example, acquiring $2t + 1$ nodes required to execute a t-Byzantine-fault-resilient critical computation on the Grid or SETI@HOME may not be a problem. However, it is desirable to reduce the overall CPU usage and network traffic due to that computation, so as to reduce the computation's performance impact felt by the regular users of those nodes.

While existing Byzantine-fault-tolerant state machine replication protocols (e.g., [1, 2, 5]) require all replicas to process requests in all protocols runs (irrespective of whether the runs are fault-free or with faults), our approach requires only a subset of the replicas, called the *primary committee*, to process requests in fault-free runs; hence our approach is *parsimonious*. When any primary committee member exhibits faulty behavior, a reconfiguration of the group occurs, resulting in a reselection of the primary committee. We do not rely on a group membership service to remove the faulty primary committee members from the replication group in order to make progress.

2 The Algorithm

We consider a client-server system in which there are two kinds of processes: a trusted client called the *gateway* (similar to a real-world corporate gateway) and n server replicas (denoted by the set $\mathcal{S}$) that implement deterministic state machines. The replicas start with the same initial state. The gateway and server replicas run on different nodes in an asynchronous distributed system. Communication between the processes takes place through FIFO, quasi-reliable (no message loss if the sender and receiver are correct), and asynchronous channels[1]. The gateway can be viewed as

*This research has been supported by DARPA contract F30602-00-C-0172.

[1] Indeed, our algorithm can be extended to handle unreliable channels that discard or reorder messages, but our assumptions allow us to avoid a discussion (about retransmissions and negative acks) that is not central to the contribution of the paper.

a serializing mechanism that accepts requests from multiple clients, enforces a total order among their requests, and forwards the requests to all the server replicas in the same order. Our algorithm complements Yin et al.'s work [5], in that our algorithm can be deployed in their execution cluster, which consists of replicas that actually execute the totally ordered requests coming from the agreement cluster (which is the conceptual equivalent of our gateway).

Correct replicas follow their specifications, while faulty ones can behave arbitrarily. A computationally bound adversary can coordinate faulty replicas in arbitrary ways. The minimum number of replicas needed to tolerate t Byzantine faults among the replicas in such a system is $n = 2t + 1$. All messages sent in the algorithm are digitally signed by the sender using public-key cryptography. If a message recipient is unable to verify the signature on the message, it simply discards the message.

Any replication algorithm is required to guarantee safety and termination. Termination ensures that if the gateway sends a request, it eventually *accepts* a response. Safety ensures that (1) at any two correct replicas i and j, the x^{th} updates to their internal states as a result of processing the x^{th} gateway requests are the same (*total order*), (2) any correct replica executes the x^{th} update to its internal state at most once, and only if the gateway sent the x^{th} request (*update integrity*), and (3) for any response corresponding to the x^{th} request *accepted* by the gateway, at least one correct replica sent that response (*response integrity*). Our algorithm does not rely on synchrony assumptions for safety. However, we rely on synchrony assumptions for liveness.

In addition to the usual safety and liveness properties, our algorithm also guarantees *parsimony*[2]. This property characterizes the reduced processing and communication activity involved during fault-free behavior of our algorithm, and distinguishes our algorithm from other Byzantine-fault-tolerant state machine replication algorithms. Informally, the property states that under normal circumstances, only the replicas in some $(t + 1)$-subset of $\mathcal{S}$ process the gateway request and send the state update to other replicas and the response to the gateway. We call this $(t + 1)$-subset the *primary committee* and the other replicas the *backups*. $(t + 1)$ is the minimum number required to avoid the case in which the committee consists of all Byzantine replicas that faithfully send the required messages to other processes but carry out the wrong processing, resulting in wrong responses to gateway requests and bad updates to backups.

Suppose that $\mathcal{PC}$ denotes the set whose elements are $(t + 1)$-subsets of $\mathcal{S}$ that could constitute primary committees. Consider two correct replicas i and j such that $i \in \mathcal{P} \wedge i \notin \mathcal{Q}$, $j \in \mathcal{Q} \wedge j \notin \mathcal{P}$, and $\mathcal{P}$, $\mathcal{Q} \in \mathcal{PC}$. Then, the parsimony property could be formally stated as follows: If both i and j process the same gateway request, then at least one of the following is true: (1) at least one replica in $\mathcal{P}$ exhibited a commission fault[3], or is suspected by $t + 1$ replicas of having exhibited an omission fault[4], and/or (2) at least one replica in $\mathcal{Q}$ exhibited a commission fault, or is suspected by $t + 1$ replicas of having exhibited an omission fault.

All replicas are initialized with the same primary committee. The gateway sends a request message to all replicas. The message carries a sequence number and the state machine operation to be executed. Request messages arrive at all replicas in the same order without any gaps. Each replica maintains its own *receive queue* for incoming messages from the gateway, and a *handled set* that contains the requests that have been processed. A committee member executes requests in sequence number order. After executing the operation indicated by a gateway request, a committee member generates a reply message and an update message. The reply message carries the result r of executing the operation requested by the gateway. The update message carries the update u to the replica's state as a result of processing the request. A committee member sends the reply message to the gateway. The committee member combines the reply and update messages in a single reply-update message which is sent to all the replicas. In the normal fault-free case, the gateway will receive reply messages from $t + 1$ replicas with identical r values. The gateway can then accept r as the result for its request message. Similarly, each replica j (whether it is a committee member or backup) will receive reply and update messages with identical r and u values (respectively) from the $t + 1$ committee members. Replica j can then perform the state update suggested by u, remove the head of the *receive queue*, and add it to the *handled set*. Then the algorithm can move on to service the next gateway request if the *receive queue* is non-empty. For simplicity, in the rest of this section, we will deal with a single gateway request that has the sequence number x.

As many as t out of the $t + 1$ committee members could be faulty. When the x^{th} gateway request is being serviced, a faulty committee member could try to disrupt the algorithm by sending bad messages (wrong r value in a reply message or wrong u value in an update message) to one or more processes. It could also refuse to send reply and/or update messages to one or more processes. Even a committee member that has not been corrupted by the adversary

[2] One can view our algorithm as an adaptation of the parsimony principle used in crash-tolerant semi-passive replication [3] to the Byzantine fault model. Semi-passive replication is a variant of the primary-backup approach, which does not rely on a group membership service to make progress in the presence of a faulty primary.

[3] A commission fault occurs when a replica sends a message that it should not have sent according to the specifications of the algorithm.

[4] An omission fault occurs when a replica does not send a required message.

may be experiencing transient (CPU or network) load conditions that prevent its reply and/or update messages from being received in time. In such cases, a reconfiguration protocol has to be initiated to ensure liveness. At the same time, a faulty replica must not be able to initiate the reconfiguration protocol when the committee indeed consists of correct members. Hence, our algorithm allows the reconfiguration protocol to be initiated by a replica only through a *reselect* message that carries proper "justification." A valid reselect message that has proper justification will convince a recipient of the message that at least one committee member has exhibited a commission fault or is suspected by $t + 1$ replicas of having exhibited an omission fault. The justification for a commission fault must be one of the three types of suspect-commission messages, and the justification for an omission fault must be suspect-omission messages from $t + 1$ replicas (described below). The convinced recipient will send its own reselect message to other replicas.

A replica j will suspect a committee member i of having exhibited an omission fault if j has not received i's reply-update message for the x^{th} gateway request even after j's local timer, which was started when the x^{th} request became the head of j's *receive queue*, expired. Replica j conveys its suspicion of i to other replicas through a suspect-omission message indicating replica i and request x. If there is a replica k that indeed received i's reply-update message for the x^{th} request in a timely fashion, then k simply forwards i's reply-update message to j upon receiving j's suspect-omission message for i. This strategy ensures that (1) if at least one correct replica has received i's reply-update message in a timely fashion, then all correct replicas will eventually receive i's message, and (2) if no correct replica has received i's reply-update message in a timely fashion, then the reconfiguration protocol will be initiated.

A committee member i can exhibit a commission fault by sending an improperly formed message with a valid signature or by sending a reply-update message with the wrong r or u values to all or a subset of the replicas. In the first case, a recipient of the improper message will send a type-1 suspect-commission message that carries i's improperly formed message. In the second case, when a replica j receives reply-update messages with different values for the result or state update from different committee members for the x^{th} gateway request, replica j will recognize that some committee member is faulty; however, it will not know which committee member is the faulty one, unless j is itself a committee member, has processed the x^{th} request, and hence knows the correct result and state update values. Replica j sends a type-2 suspect-commission message that carries reply-update messages from two or more committee members with differing values for the result and/or state update as the justification. A recipient replica k of j's type-2 suspect-commission message will not know which committee member is faulty, but will be convinced that the reconfiguration protocol must be initiated, as at least one committee member has exhibited a commission fault.

The reconfiguration protocol consists of two stages. In the first stage, all replicas temporarily become active replicas, service the x^{th} gateway request, send the reply messages to the gateway, and exchange reply-update messages among themselves. A replica moves to the second stage after it has received reply-update messages with identical result and state update values from $t + 1$ replicas. That will eventually happen, since there are at least $t + 1$ correct replicas. At the end of the first stage, previous committee members that sent wrong result and/or state update values in their reply-update messages can be easily identified. For each such faulty replica, a type-3 suspect-commission message will be sent carrying the replica's "bad" reply-update message and the "good" reply-update messages from $t+1$ other replicas. In the second stage of the reconfiguration protocol, a reselection of the primary committee occurs. Once committee reselection is completed, replicas that are not members of the new committee become backups and do not service the next gateway request. If some members of the new committee exhibit faulty behavior, then the reconfiguration protocol will be initiated again, and so on.

Suppose that $\mathcal{PC}$ consists of all the $t+1$-subsets of $\mathcal{S}$. Then, $|\mathcal{PC}| = C^{2t+1}_{t+1}$. Let $\mathcal{PC}_{ord}$ be an ordered list containing all the elements of $\mathcal{PC}$. $\mathcal{PC}_{ord}$ is known to all replicas. Each replica maintains a local integer called the *committee number*. If the committee number has a value g, then the g^{th} element in $\mathcal{PC}_{ord}$ represents the primary committee. All replicas start with committee number 1. Each replica j maintains an omission-fault list and a commission-fault list. A replica i is added to j's omission-fault list after j has seen suspect-omission messages from $t + 1$ replicas (possibly including j itself) for i. Replica i is added to j's commission-fault list after j has seen a type-1 or type-3 suspect-commission message (possibly from itself) for i. At a correct replica j, if the committee number is g at the start of the reconfiguration protocol, then the committee number is advanced to the smallest integer $\overline{g}$ greater than g, such that the $\overline{g}^{\text{th}}$ element in $\mathcal{PC}_{ord}$ does not contain any replicas in j's omission-fault list or commission-fault list. Now, the $\overline{g}^{\text{th}}$ element in $\mathcal{PC}_{ord}$ is the new primary committee from j's perspective. Correct replicas may temporarily differ in their perspectives of the primary committee. However, their perspectives will eventually concur, since their fault lists will eventually become the same. That is ensured because a correct replica sends a reselect message with proper justification upon any addition to its fault lists. A convinced recipient will update its own fault lists, and send its own reselect message with proper justification.

If the gateway has not accepted a result within a certain time after sending its request, it retransmits the request to all the replicas. If any correct replica has received reply-update messages with identical result and state-update values from $t + 1$ replicas for that request, then the replica simply forwards the reply messages to the gateway. Otherwise, the reconfiguration protocol will eventually be initiated, causing all correct replicas to process the request and send the reply message to the gateway. In either case, the gateway is guaranteed to accept a response eventually.

3 Discussion

Previous Byzantine-fault-tolerant state machine replication algorithms did not distinguish between fault-free and faulty operation. Their principle was active replication in which all correct replicas always process all requests. Our algorithm obtains a reduction of processing and communication activity in the normal fault-free case by using only $t + 1$ out of $2t + 1$ replicas (the primary committee) to process the client request. In the presence of faulty committee members, a reconfiguration protocol is initiated, which causes all replicas to process the request (as in active replication). When there is a faulty committee member, the additional overhead imposed by the algorithm (relative to active replication) consists of the fault detection latency. However, this is a small price to pay considering the improved performance obtained in the fault-free case, in which the system is expected to spend most of its runtime.

Our algorithm is not just a simple hybrid protocol that uses a smaller subset of replicas in the fault-free case and switches to using all the replicas in the faulty case. Only faults in the primary committee can initiate the switch. Also, faulty backups cannot force reconfiguration when the committee members are functioning properly (because the reselect messages require proper justification). The eventual goal of the reconfiguration protocol is to make the primary committee "settle down" at a $(t+1)$-subset of $\mathcal{S}$ consisting of all correct replicas. Though there are C^{2t+1}_{t+1} elements of $\mathcal{PC}_{ord}$, the number of times the reconfiguration protocol has to be initiated for the primary committee to settle down on an all-correct $(t + 1)$-subset is only $O(f)$, where $f \leq t$ is the actual number of faults. The reason is that while the next primary committee is being chosen, the elements of $\mathcal{PC}_{ord}$ that contain any replicas in the commission-fault list or the omission-fault list are skipped over.

Our algorithm is safe under denial-of-service (DoS) attacks. A DoS attack may delay the gateway's acceptance of a correct response, but it will not succeed in making the gateway accept a wrong response. The attack may also cause non-corrupted replicas to be added to the omission-fault list. As a result, the number of replicas in either the omission-fault list or commission-fault list may exceed t when a DoS attack is underway. When that happens, the reconfiguration protocol is initiated as usual; however, in the second stage of the reconfiguration protocol, entries in the omission-fault list are refreshed, and the committee number is reset to the smallest integer $\hat{g}$ greater than or equal to 1, such that the $\hat{g}^{\text{th}}$ element in $\mathcal{PC}_{ord}$ does not contain any replicas in the commission-fault list. Such a refresh of the omission-fault list may happen several times during the DoS attack. Obviously, the $O(f)$ bound mentioned above will not hold during such periods of instability in the system. However, as long as no more than t replicas have been actually *corrupted* by the adversary, the algorithm will make progress after the DoS attack is over.

In our algorithm, backups do not actually process the requests; they are required only to monitor the progress of the protocol, update their replicated states based on the reply-update messages from the committee members, and generate suspect messages if a committee member is not behaving correctly. For some applications, updating the state may be a very low portion of the processing load (e.g., read-only file systems). However, for applications in which updating the state forms a significant part of the processing load (e.g., read-write file systems), this strategy may not be sufficient to significantly reduce the processing load at backups in the fault-free case. In addressing this issue, we observe that the state of a backup need not be up to date after every request, but only just before it starts processing the gateway request during the first stage of the reconfiguration protocol. This suggests that state updates can be performed at backups in a lazy (on-demand) fashion. In [4], we describe an extension to the algorithm that uses checkpointing to perform lazy updates at backups. [4] also describes a modified version of the algorithm that uses MACs instead of digital signatures during normal operation for the reply-update messages.

Acknowledgments We thank Christian Cachin for his helpful comments, and Jenny Applequist for her editorial help.

References

[1] C. Cachin and J. A. Poritz, "Secure Intrusion-Tolerant Replication on the Internet," In Proc. DSN-2002, pp. 167-176, 2002.

[2] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery," ACM TOCS, 20(4):398–461, Nov. 2002.

[3] X. Defago, A. Schiper, and N. Sergent, "Semi-Passive Replication," In Proc. SRDS-17, pp. 43-50, Oct. 1998.

[4] H. V. Ramasamy, A. Agbaria, and W. H. Sanders, "Intrusion-Tolerant Parsimonious State Machine Replication," University of Illinois Coordinated Science Laboratory technical report, to appear.

[5] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating Agreement from Execution for Byzantine Fault Tolerant Services," In Proc. SOSP-2003, pp. 253-267, 2003.