Byzantine Agreement with a Strong Adversary in Polynomial Expected Time

Valerie King Jared Saia

Department of Computer Science, University of Victoria, Canada, val@cs.uvic.ca, +1 250-472-5727
Department of Computer Science, University of New Mexico, USA, saia@cs.unm.edu

PODC Brief Announcement Submission

Abstract

In a paper appearing in STOC 2013, we considered Byzantine agreement in the classic asynchronous message-passing model. The adversary is *adaptive*: it can determine which processors to corrupt and what strategy these processors should use as the algorithm proceeds. Communication is *asynchronous*: the scheduling of the delivery of messages is set by the adversary, so that the delays are unpredictable to the algorithm. Finally, the adversary has *full information*: it knows the states of all processors at any time, and is assumed to be computationally unbounded. Such an adversary is also known as "strong". We presented the first known polynomial expected time algorithm to solve asynchronous Byzantine Agreement when the adversary controls a constant fraction of processors. This is the first improvement in running time for this problem since Ben-Or's exponential expected time solution in 1983.

How can we build a reliable system our of unreliable parts? Byzantine agreement is fundamental to addressing this question. The Byzantine agreement problem is to devise an algorithm so that n agents, each with an private input can agree on a single common output that is equal to some agent's input. For example, if all processors start with 1, they must all decide on 1. The processors should successfully terminate despite the presence of $t = \theta(n)$ bad processors. An adversary controls the behavior of the bad processors which can deviate from the algorithm in arbitrary ways. Byzantine agreement is one of the most fundamental problems in distributed computing. Studied for over 30 years, it is referenced in thousands of papers.

In a paper presented at STOC 2013, we considered Byzantine agreement in the challenging classic asynchronous model. The adversary is *adaptive*: it can determine which processors to corrupt and what strategy these processors should use as the algorithm proceeds. Communication is *asynchronous*: the scheduling of the delivery of messages is set by the adversary, so that the delays are unpredictable to the algorithm. Finally, the adversary has *full information*: it knows the states of all processors at any time, and is assumed to be computationally unbounded. Such an adversary is also known as "strong" [3].

The major constraint on the adversary is that it cannot predict future coinflips, and we assume that each processor has its own fair coin and may at any time flip the coin and decide what to do next based on the outcome of the flip.

Time in this model is defined to be the maximum length of any chain of messages (see [3, 7]). In particular, all computation by individual processors is assumed to be instantaneous, and sending a message over the network is counted as taking 1 unit of time.

The only previously known results for this classic model are the works of Ben-Or (1983) [5] and Bracha (1984) [4]. Ben-Or gave a Byzantine agreement algorithm tolerating $t < n/5$. Bracha improved this tolerance to $t < n/3$. Unfortunately, both of these algorithms run in exponential expected time if $t = \Theta(n)$. We presented the first algorithm for this problem to achieve better than exponential expected run time. Our main result is the following.

Theorem 1. *Let n be the number of processors. There is a $t = \Theta(n)$ such that Byzantine Agreement can be solved in expected time $O(n^{2.5})$ and expected polynomial bits of communication, in the asynchronous message passing model with an adaptive, full-information adversary that controls up to t processors.*

Note that we leave open the problem of whether the computations required by each individual processor can be done in polynomial time.

Technical Overview We start with Ben-Or's 1983 algorithm. In this algorithm, roughly speaking, the processors take the majority of each others' votes. When there is a sufficiently large majority of processors which agree on the same bit, the adversary cannot affect the outcome. When there is not a sufficiently large majority, some or all of the processors flip their coins. If the processors which flip their coins happen to all flip them to the same value and that value happens to agree with the value held by the processors which do not flip, the algorithm terminates successfully in the next round. Ben-Or's algorithm reduces Byzantine agreement to the problem of generating a mutually agreed upon coinflip. In this sense it is similar to Rabin's global coinflip algorithm [10], which however assumes the existence of a global coin.

We thus follow the technique in the consensus literature of reducing the agreement problem to the problem of producing a commonly agreed upon coinflip (see the survey [1]; we were particularly inspired by the result in [2] for producing consensus in a shared memory model). The consensus problem is equivalent to Byzantine agreement, except that processors taken over by the adversary suffer crash faults and thus no longer send out messages.

In the consensus problem, a prevalent technique for generating a common coin is to have the processors generate and send out many coinflips. If the bias of these coinflips is sufficiently high, then if each processor takes the majority value of the coinflips, 1) all processors will obtain the same value; and 2) that value will be an unbiased coinflip. It suffices to generate n^2 individual coinflips to ensure that with constant probability, the bias generated by these coinflips will exceed the $O(n)$ bias that the adversary can introduce through scheduling of messages and crash faults.

Unfortunately, in Byzantine agreement, the adversary can introduce $\Theta(nt) = \Theta(n^2)$ bias through adversarily determined "coinflips". Thus, we need a new technique for limiting adversarial bias. Our basic approach is for each processor to run Ben-Or's algorithm multiple times and, for each processor p to add processors with suspiciously large bias to its "suspect list" so that their coinflips are subsequently ignored by p.

In particular, over enough iterations of Ben-Or's algorithm, if no decision has occurred, there must be a group of t or fewer processors which have produced coinflips with a suspiciously high amount of bias. These highly biased coinflips must have been received by enough good processors to prevent a decision. We show that a good processor can use information about these coinflips to ensure that its suspect list has the following properties. First, if Ben-Or's fails over many iterations, eventually all bad processors will be added to the suspect list. Second, no more than t good processors are ever added to the suspect list.

When all bad processors are added to each good processor's suspect list, agreement is reached within an expected constant number of iterations of Ben-Or's algorithm. One technical challenge is to show that not too many good processors are added to the suspect list of any good processor.

Related Work Randomized algorithms for Byzantine agreement with resilience $t = \Theta(n)$ and constant expected time have been known for 25 years, under the assumption of private channels (so that the adversary cannot see messages passed between processors) [8], or cryptographic assumptions. Under these kind of assumptions, more recent work shows that optimal resilience can be achieved [6].

Byzantine agreement was also more recently shown to require only polylogarithmic time in the full information model if the adversary is *static*, meaning that the adversary must choose the faulty processors at the start, without knowing the random bits of the algorithm [9]. The technique is to elect a very small subset of processors which contain a less than 1/3 fraction of faulty processors; this subset then runs the exponential time algorithm on their inputs. Such a technique does not seem applicable when the adversary is adaptive and can decide to corrupt the elected set after it sees the result of the election.

References

[1] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16:165–175, 2003.

[2] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. *J. ACM*, 55(5), 2008.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*, page 14. John Wiley Interscience, 2004.

[4] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *The First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[5] M. Ben-Or. Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM New York, NY, USA, 1983.

[6] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, 1993.

[7] B. Chor and C. Dwork. Randomization in Byzantine agreement. *Advances in Computing Research*, 5:443–498, 1989.

[8] P. Feldman and S. Micali. Byzantine agreement in constant expected time (and trusting no one). In *FOCS*, pages 267–276, 1985.

[9] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Scalable algorithms for byzantine agreement and leader election with full information. *ACM Transactions on Algorithms(TALG)*, 2009.

[10] M. Rabin. Randomized byzantine generals. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 403–409, 1983.