An O(lg n) Expected Rounds Probabilistic Byzantine Generals Algorithm

(The Bigger They Are, The Harder They Fall)

Gabriel Bracha

TR 84-631
August 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

*Partial support for this work was provided by NSF grant No. 83-03135.

An O(lg n) expected rounds randomized Byzantine Generals algorithm (The bigger they are, the harder they fall)

Gabriel Bracha

Cornell University
Ithaca, New York 14853

ABSTRACT

Byzantine Generals algorithms enable processes to reliably broadcast messages in a system of n processes where up to t of the processes may be faulty. The algorithms are conducted in synchronous rounds of message exchange. For a system where $n = (3+\delta)t$ we prove the existence of a randomized algorithm whose expected number of rounds is $O(lg\ n)$. This is an improvement on the lower bound of $t+1$ rounds required for deterministic algorithms and on the previous result of $t / lg\ n$ expected number of rounds for randomized algorithms.

*Partial support for this work was provided by NSF grant No. 83-03135.

1. Introduction

One of the fundamental problems in distributed systems is achieving agreement among processes when some of the processes are faulty. The *Consensus* and *Byzantine Generals* problems are two classical paradigms of agreement problems that have been extensively studied in recent years.

In a consensus protocol, each process p starts with an initial $value_p$, and the processes try to reach a consensus value. The algorithm terminates when each correct process has made an irreversible *decision* on some value. The protocol must satisfy the following properties:

Bivalence: if all the correct processes start with the same value v then they all decide v.

Agreement: all the correct processes decide on the same value.

In a Byzantine Generals protocol one process, the *transmitter*, tries to broadcast its value, v, to the rest of the processes. The algorithm has to satisfy the following properties:

Validity: if the transmitter is correct all the correct processes decide on v.

Agreement: all the correct processes must agree on the same value.

The two problems are equivalent. Given a Byzantine Generals algorithm one can achieve consensus by the following: First, each process broadcasts its value via the Byzantine Generals algorithm. Then, each process takes the majority of the values received. Given a consensus protocol one can obtain a Byzantine Generals algorithm by the following: First, the general sends its value to all the processes. Then, the processes run a consensus protocol on the received values.

The typical setting for solving these problems is a system of n processes; t of the processes are *faulty* and may deviate from the algorithm, the rest of the processes are *correct*. An algorithm on the system is conducted in synchronized rounds. In each round a process sends a set of messages to the other processes, receives a set of messages, and performs some local computation. The algorithms is called *t-resilient* if it remains valid in the presence of up to t faulty processes.

There is a whole spectrum of failure types according to the degree the faulty processes can deviate from the algorithm [Hazd83]. In this paper we are concerned with *Byzantine failures*, where faulty processes can send spurious and contradictory messages. In order to allow for the worst possible behaviour of the algorithm we even assume that the faulty processes play an *adversary* role against the correct processes, and use their joint "knowledge of the system" to prevent agreement. The knowledge of the system available to the faulty processes is an extremely important but subtle parameter of the problem. In this paper we assume either of the following :

1. Faulty processes have no knowledge about the communication between correct processes (completely connected network).

2. Faulty processes have no knowledge of the contents of encrypted (e.g., RSA encrypted [Rive78]) messages.

Much attention has been given to the number of rounds necessary to reach agreement. For deterministic protocols a lower bound of $t + 1$ was proved [Dole82b, Fisc82a, Lamp82a] for both problems and all failure types. Algorithms meeting this bound are presented in [Dole81, Dole82a, Dole82b, Lamp82a, Lamp82b].

Probabilistic algorithm were presented in [BenO83, Brac83, Brac84a, Chor84, Rabi83]. In these algorithms, the number of rounds to reach termination is not bounded. Instead, the algorithms have the following *termination* property:

$$\lim_{k \to \infty} \text{Probability [some correct process has not decided within } k \text{ rounds]} = 0.$$

Thus, though the number of rounds is not bounded, the probability that the algorithm does not terminate is 0. For this type of algorithm our complexity measure is the expected number of rounds to reach agreement. The algorithms of [Brac84] and [BenO83] have the following interesting property: if $t = O(n)$ then the expected number of rounds is exponential in n; but if $t = O(\sqrt{n})$ then the expected number of round to reach agreement is a constant. In [Chor84] a randomized algorithm is presented with $t / lg\ n$ expected number of rounds and no assumptions are made on the knowledge available to the faulty processes. The model of [Rabi83] is different since it assumes a reliable initial distribution of a pre-randomized sequence of coin tosses. In that model the expected number of rounds is 4.

In this paper we show the existence of a t-resilient consensus (Byzantine Generals) algorithm for Byzantine processes such that, for $n = (3+\delta)t$ and $\delta > 0$, the expected number of rounds is $O(lg\ n)$. The algorithm is based on the algorithms of [BenO83] and [Brac84]. The superior performance of the algorithm depends on the existence of a family of subsets of processes that satisfies certain requirements. We prove that the desired family exists but we are unable to present it.

2. Algorithm outline

The algorithm is based on the randomized algorithms of [BenO83] and [Brac83]. These algorithms have the property that if the number of the faulty processes is $O(\sqrt{n})$ then the expected running time is constant. However if the number of faulty processes is $O(n)$ then the running time is exponential in n.

The essential idea is to reduce the proportion of faulty processes from $n/t > 3$ down to $O(1/\sqrt{n})$. We accomplish this reduction by creating *compound virtual processes*. Then, we run an algorithm similar to those in [BenO83, Brac83] on the compound processes. Each compound process P is a set of s *actual processes* $\{p_1, \ldots, p_s\}$ that are P's *components* or *sub-processes*. The compound processes can overlap in their sub-processes. Compound processes implement each basic step of the algorithm by running a consensus algorithm among their sub-processes. We define a compound process to be faulty if more than a third of its sub-processes are faulty.

The choice of s, the *size* of the compound processes, and of m, their number, is crucial to the performance of the algorithm. If $s > 3t$ then all the compound processes are correct, but each basic step will take $t + 1$ rounds, and there is no improvement in the running time. If $s = 1$ then each basic step will take only one round, but the proportion of faulty compound processes will be the same as the proportion of actual faulty processes, i.e., the algorithm will require an exponential number of rounds.

We show that for $s = O(lg\ n)$ and for $m = O(n^2)$ there is a fixed way of constructing m compound processes of size s such that less than $\sqrt{m}$ of them are faulty. Given these processes, the expected number of "virtual rounds" required for running the algorithm on the compound processes is constant. Each "virtual round" requires reaching consensus among the s sub-processes and takes $O(s) = O(lg\ n)$ rounds. Thus, the expected running time of the algorithm is $O(lg\ n)$.

The translation of algorithms from actual processes to compound processes does not depend on the algorithm we use to implement the compound processes or on the algorithm that the compound processes are intended to run. Thus, we can plug different consensus algorithms into this framework, either to implement compound processes, or to achieve consensus among the compound processes.

3. The basic algorithm

To achieve consensus among the compound processes we can use a simplified version of the $n/5$-resilient algorithm in [BenO83]. The algorithm is conducted in phases, each phase consists of three rounds of messages exchange. For simplicity of presentation the algorithm described here does not halt once the processes decide.

Algorithm 1.

phase(i): (By process p)

1. *Send* $Value_p$ to all the processes.
2. If *Received* more than $(n + t)/2$ messages with the same value v then $Value_p := v$ and *Send* $Value_p$ to all the processes.
3. (i) If *Received* † more than $(n + t)/2$ messages with value v then $Decision_p := v$.
   (ii) If *Received* at least $t + 1$ messages with value v then $Value_p := v$.
   (iii) Otherwise, $Value_p :=$ *Coin_Toss* (0 or 1 with probability ½).
   (Go to round 1 of phase $i + 1$.)

Figure 1. The basic t-resilient consensus algorithm for n processes.

Theorem 1. For $t \leq n/5$, Algorithm 1 is a t-resilient consensus algorithm.

Proof: The proof is identical to that of [BenO83] and is omitted. □

Theorem 2. If $t = O(\sqrt{n})$ then the expected number of rounds to reach agreement in Algorithm 1 is constant.

† *Receive* messages that were sent at step 2 of phase i.

Proof : The proof is identical to those of [BenO83, Brac83] and is omitted. □

Consider now a system with t faulty processes, $(n-t)$ correct processes, and d *spectator* processes that do not send any messages at all, but change their state according to the algorithm (and the messages they receive).

Corollary 1. Both Theorem 1 and Theorem 2 hold for a system with $t < n/5$ faulty processes, $(n-t)$ correct processes, and d spectator processes.

Proof: The spectator processes do not affect the progress of the system during the algorithm. Therefore, Theorem 1 and Theorem 2 apply to the n participating processes. However, the spectator processes receive sets of messages that any of the participating processes could have received, and therefore they must decide on the same value at the same round as a correct process would, i.e, Theorem 1 and Theorem 2 apply to them too. □

4. Implementation of compound processes

In this section we show how to implement the compound processes. More specifically, we show how a compound process computes and how it sends and receives messages. That has to be done in a way that is consistent with the semantics of these actions. We also define what constitutes a correct or faulty compound process, and prove that our definitions are consistent with the notion of failure and correctness in our system.

In this section we adopt the following convention: lower-case letters are used to denote the "actual world" (processes, primitives, variables, and attributes), while upper-case letters are used to describe the "virtual world". Thus, *Send* is an abstract operation with the semantics one would expect from its name; *send* is a low level realization of *Send* on actual processes (e.g., hardware and communication protocol), and *SEND* is an implementation of *Send* on compound processes that is accomplished by *send.*

4.1. Variables, attributes and primitives of compound processes

Let $P = \{p_1, \ldots, p_s\}$ be a compound process. The "virtual" local variable VAR of process P is denoted as VAR_P. VAR_P is implemented by a set of local variables $var_{P,1}, \ldots, var_{P,s}$ in $p_1, \ldots, p_s$ respectively. The value of VAR_P is defined to be v if v is the value of more than $2s/3$ of $var_{P,1}, \ldots, var_{P,s}$. A process P is $FAULTY$ if at least $s/3$ out of $\{p_1, \ldots, p_s\}$ are *faulty*, otherwise P is *CORRECT*.

In the implementation we use the functional notation *Byzantine*($var_{P,i}$) to indicate that all of P's sub-processes are executing a Byzantine Generals algorithm to determine the value of variable *var* of p_i. More specifically:

Byzantine($var_{P,i}$):

If p_i is *correct*, then return $var_{P,i}$ for every p_j

If p_i is *faulty* but P is *CORRECT*, then return the same value,
either $var_{P,i}$ or *undefined*, to every *correct* p_j.

If both p_i is *faulty* and P is *FAULTY*, then return either $var_{P,i}$ or *undefined*.

Byzantine can be implemented as in [Dole81] where it requires $s/3+1$ rounds, or in any other way. Given *Byzantine* we can define

Consensus(VAR_P):

If $\underset{1 \le k \le s}{majority}(\, Byzantine(var_{P,k})\,) = v$ and $v \neq undefined$ then return v

Otherwise return $var_{P,j}$ to p_j.

We further define

Flip(P):

If P is *CORRECT*,then return the same value, 0 or 1 with equal probability,
to every *correct* p_j.

If P is *FAULTY*, then return 0 or 1.

Algorithms for *Flip* are presented in [Brod83] and [Yao83]. The algorithm in [Yao83] takes $s/3$ rounds. However it assumes that faulty processes cannot "know" the messages sent between correct processes. The algorithm in [Brod83] takes s rounds; it involves an ingenious use of public key cryptography [Rive78] and secret sharing [Sham79]. The algorithm requires some cryptographic assumptions whose details are specified in the Appendix.

Let $P = \{p_1, \ldots, p_s\}$ and $Q = \{q_1, \ldots, q_s\}$ be compound processes. We define the following primitives for P and Q:

INITIALIZATION

for all $1 \le i \le s$ $value_{P,i} := value_{p_i}$;

$value_{P,i} := Consensus(VALUE_P)$.

Deterministic computation: $VAR_P := f(PAR_P^1, \ldots, PAR_P^k)$

for all $1 \le i \le s$ $var_{P,i} := f(par_{P,i}^1, \ldots, par_{P,i}^k)$.

Probabilistic computation: $VAR_P := COIN\_TOSS$

for all $1 \le i \le s$ $var_{P,i} := Flip(P)$

P *SEND*s m to Q

for all $1 \le i,j \le 1$ p_i *send* (P,m) to q_j

P *RECEIVE*s m from Q

for all $1 \le i \le s$ **if** p_i *receive*d more than $2s/3$ messages (Q,m)

then $msg(Q)_{P,i} := m$

p_i *receives* $Consensus(msg(Q)_P)$

4.2. A consensus algorithm for compound processes

The definitions of the previous section give rise to Algorithm 2, a version of designated for compound processes.

4.3. Correctness proof of the implementation

For a fully general and formal proof we have to provide an axiomatic description of our system, i.e., axiomatize variables, statement, processes (correct and faulty) and message passing. Then we have to show that our implementation provides a "homomorphism" between the actual and the virtual systems that preserves all the axioms. This task is beyond the scope of this paper; so, we restrict ourselves to the context of , and use some less formal arguments.

Algorithm 2.

phase(I): (By process P)

1. *Send* $VALUE_P$ to all the processes.

2. If $RECEIVE$d more than $(N+T)/2$ messages with the same value v then $VALUE_P := v$ and $SEND$ $VALUE_P$ to all the processes.

3. (i) If $RECEIVE$d more than $(N+T)/2$ messages with value v then $DECISION_P := v$.

   (ii) If $RECEIVE$d at least $T+1$ messages with value v then $VALUE_P := v$.

   (iii) Otherwise, $VALUE_P := COIN\_TOSS$

   (Go to round 1 of phase $I+1$.)

Figure 2. A T-resilient consensus algorithm for N compound processes.

Theorem 3. For $T < N/5$, Algorithm 2 is a T-resilient consensus algorithm for compound processes.

Proof: Let $P_1, \ldots, P_N$ be N compound processes such that $T < N/5$ of them are $FAULTY$, and let $P = \{p_1, \ldots, p_s\}$ be a $CORRECT$ process. The use of *Byzantine*, *consensus*, and *Flip* in the primitives guarantees that all P's *correct* sub-processes are in the same state after executing each primitive. Since more than $2s/3$ of p's sub-processes are *correct* the value of VAR_P is that of $var_{P,i}$ for any p_i *correct.* Every message $RECEIVE$d from P is a message sent by all its *correct* sub-processes. Furthermore, $COIN\_TOSS$ produces a truly random bit. Thus, P behaves exactly as any of its *correct* sub-processes.

On the other hand, a $FAULTY$ process Q can send different messages to different processes, and fix the value of its "coin". Thus Q behaves like an actual *faulty* process $\bar{q}$ that *sent* the messages that were $RECEIVE$d from Q.

Thus, a system of N compound processes behaves under Algorithm 2 exactly as an *induced system* of N actual processes under Algorithm 1. A $CORRECT$ process is represented in the induced system by any of its *correct* sub-processes. A $FAULTY$ process Q is represented by the previously described *faulty* process $\bar{q}$. Suppose that running algorithm 2 on $P_1, \ldots, P_N$ violates the agreement, bivalence or termination requirement. Then these requirements are also violated by running Algorithm 1 on the induced system, hence a contradiction. □

Theorem 4. If $T = O(\sqrt{N})$ then the expected number of rounds to reach agreement in running algorithm 2 on $P_1, \ldots, P_N$ is constant.

Proof : Note that *COIN_TOSS* produces a random bit at all the *correct* sub-processes of a *CORRECT* compound process. Consider now the induced system as in Theorem 3. Agreement is reached in the original system if and only if it is reached in the induced system at the same round, i.e., in both systems the number of rounds of rounds to reach agreement has the same distribution. By Theorem 2, the expected number of rounds to reach agreement in the induced system is constant, and therefore the expected number of rounds to reach agreement in the original system is the same. □

Running Algorithm 2 on $P_1, \ldots, P_N$ can also be viewed as running a consensus algorithm on $p_1, \ldots, p_n$ since each actual process p_i, while participating as a sub-process, receives messages, undergoes state transitions, and can decide on a value.

Theorem 5. Given compound processes $P_1, \ldots, P_N$ that are composed of sub-processes $p_1, \ldots, p_n$, such that $T < N/5$, then Algorithm 2 is a also consensus algorithm for $p_1, \ldots, p_n$.

Proof: Let p be an actual process that belongs as a sub-process to $P_1, \ldots, P_k$. We can regard p as k distinct virtual processes $p^1, \ldots, p^k$. Because of the use of *Consensus* in the implementation of *RECEIVE* each p^j can *RECEIVE* all the messages, but it cannot *SEND* any. Thus, any execution of Algorithm 2 on $P_1, \ldots, P_n$ can be viewed as an execution of Algorithm 2 on $P_1, \ldots, P_N, p_1^1, \ldots, p_1^{k_1}, \ldots, p_n^1, \ldots, p_n^{k_n}$, where $p_1^1, \ldots, p_n^{k_n}$ participate as spectator compound processes of size one. By Corollary 1, Algorithm 1 is a consensus protocol for the induced system and therefore Algorithm 2 is a consensus protocol on the compound system. This immediately guarantees the agreement and termination properties. To ensure bivalence we observe that if all the correct sub-processes start with the same initial value v, then after *INITIALIZATION*, v is the *VALUE* of all the *CORRECT* processes. By the bivalence of Algorithm 2 all the *CORRECT* compound processes decide v. By Corollary 1, all the correct spectator processes decide v too, i.e, the *correct* processes in $p_1, \ldots, p_n$ decide v. □

Note that the translation of the basic algorithm to compound processes depends entirely on the primitives used in the algorithm (*Send, Receive, Coin_Toss*, etc.) and does not depend at all on which algorithm we use to implement *Byzantine* or on which consensus algorithm the compound processes execute.

5. Choosing m and s

Let n, t, s, and m be as discussed in the algorithm outline. We further assume that $n/t = 3+\delta$ for some constant $\delta > 0$. Define

$$B(s,t) = \sum_{s/3 \le k \le s} \frac{\binom{s}{k} t^k (n-t)^{s-k}}{n^s}$$, the tail of the binomial distribution.

Lemma 1. Let $s < t$ and $n/t = 3+\delta$. Then $lg\ B(s,t) < -\frac{s\epsilon}{3} + lg\ n$ for some constant $\epsilon > 0$.

Proof: For $k \ge s/3$, define $b_k = \frac{\binom{s}{k} t^k (n-t)^{s-k}}{n^s}$.

Consider $\frac{b_k}{b_{k+1}} = \frac{k+1}{s-k} \cdot \frac{n-t}{t}$.

$k \ge s/3$ and therefore $k/(s-k) \ge 1/2$ and $\frac{b_k}{b_{k+1}} \ge 1+\delta/2$.

Hence, $\sum_{s/3 \le k \le s} b_k$ converges faster than a geometric sequence with quotient $1/(1+\delta/2)$.

Thus, $B(s,t) = \sum_{s/3 \le k \le s} b_k < (1+2/\delta) b_{s/3}$,

and $lg\ B(s,t) \le lg\ (b_{s/3}) + lg\ (1+2/\delta)$.

To estimate $lg\ (b_{s/3})$ we use the Stirling approximation:

$$lg\ n! = n \cdot lgn - n \cdot lge + lgn/2 + lg 2\pi/2 + o(1/n).$$

Applying it to $b_{s/3} = \frac{\binom{s}{s/3} t^{s/3} (n-t)^{2s/3}}{n^s} = \binom{s}{s/3} (3+\delta)^{-s} (2+\delta)^{2s/3}$

we get: $lg\ b_{s/3} = \left[slg\ s - (s/3) lg\ (s/3) - (2s/3) lg\ (2s/3) \right]$

$+ \frac{1}{2} \left[lg\ s + lg\ s/3 - lg\ 2s/3 - lg\ \pi \right] - slg\ (3+\delta) + (2s/3) lg\ (2+\delta)$

$< -\frac{s}{3} lg \frac{4(3+\delta)^3}{27(2+\delta)^2} = \frac{s\epsilon}{3}$ for some $\epsilon > 0$.

Since $\delta \geq 3/n$, $lg(1+2/\delta) \leq lg\ n$, and, $lg\ B(s,t) < -\frac{s\epsilon}{3} + lg\ n$. □

Now we prove that there is a way of constructing m compound processes such that less than $\sqrt{m}$ of them are faulty.

Lemma 2. Given n items such that t of them are faulty, and $n/t = 3+\delta$ for some constant $\delta > 0$, then there is a sequence of m sequences of s items, such that for any configuration of the faulty items less than $\sqrt{m}$ sequences have more than $s/3$ faulty items. Also, $s = O(lg\ n)$ and $m = O(n^2)$.

Proof: Consider L a list of all the sequences of m sequences of size s, $|L| = n^{sm}$. L provides us with a canonical representation for these sequences. We can specify every sequence by its index in L, which requires $lg\ |L| = ms \cdot lg\ n$ bits. Furthermore, in any bit-string representation of these sequences, some sequence requires $lg\ |L|$ bits. Otherwise, two different sequences will map to the same bit string.

Suppose that the converse of the lemma's statement holds, i.e., for any sequence of m sequences of size s there is some configuration of faulty items such that more than $\sqrt{m}$ sequences have more than $s/3$ faulty items. Then we can specify each sequence as follows:

1. Specify A, a configuration of faulty processes.
2. Specify B, a sequence of $\sqrt{m}$ sequences of size s such that each sequence has more than $s/3$ members in A.
3. Specify C, a sequence of $m - \sqrt{m}$ sequences of size s.
4. Specify D, a way of merging B and C.

Consider the number of bits required for this representation scheme.

1. To specify the configuration of faulty items we need only n bits; the i'th bit is 1 if the i'th item is faulty.
2. Let L^A be a list of all sequences of size s that have more than $s/3$ members in A. There are $|L^A| = \sum_{s/3 \leq k \leq s} \binom{s}{k} t^k (n-t)^{s-k} = B(s,t) \cdot n^s$ of these sequences. We can specify each sequence by its index in L^A. To specify a sequence of $\sqrt{m}$ sequences in L^A we need $\sqrt{m} \cdot lg\ |L^A|$ bits. By Lemma 1,

$$\sqrt{m} \cdot lg\ |L^A| = \sqrt{m} \cdot \left(lg\ B(s,t) + s\ lg\ n\right) < \sqrt{m} \cdot \left(-\frac{s\epsilon}{3} + lg\ n + s\ lgn\right)$$

3. To specify C we need $(m - \sqrt{m}\,) \cdot s lg\ n$ bits.

4. To specify the merging of B and C we need a list $i_1, \ldots, i_{\sqrt{m}}$. The j'th sequence in B is inserted after the i_j'th sequence in C. Thus D requires $lg\ m \cdot \sqrt{m}$ bits.

Compare the number of bits in this representation with the canonical one.

$$ms \cdot lg\ n - \left[n + \sqrt{m} \cdot \left(-\frac{s\epsilon}{3} + lg\ n + s\ lgn \right) + (m - \sqrt{m}\,) \cdot s\ lgn + lg\ m \cdot \sqrt{m} \right]$$

$$= \sqrt{m} \left(\frac{s\epsilon}{3} - lg\ n - lg\ m \right) - n$$

For $m = (n+1)^2$ and $s = 9lg\ (n+1)/\epsilon$ the difference is greater than 0. This is a contradiction, since it is impossible that all $|L|$ sequences have a bit string representation shorter than $lg\ |L|$. Therefore there must exist a sequence of m sequences of size s such that for any configuration of the faulty items less than $\sqrt{m}$ of the sequence contains more than $s/3$ faulty items. □

Corollary 2. Given n items such that t of them are faulty, and $n/t = 3 + \delta$ for some constant $\delta > 0$, then for any constant k, there is a sequence of m sequences of s items, such that for any configuration of the faulty items less than $\sqrt{m}/k$ sequences more than $s/3$ faulty items. Also, $s = O(lg\ n)$ and $m = O(n^2)$.

Proof: The proof is exactly as in Lemma 2 except that we choose $m = k^2(n+1)^2$ and $s = 9lg\ (n+1)/\epsilon + 2lgk$. □.

Corollary 3. Given n items such that t of them are faulty, and $n/t = 3 + \delta$ for some constant $\delta > 0$, then for some $s = O(lg\ n)$ and $m = O(n^2)$, the probability that in a random sequence of m sequences of size s more than $\sqrt{m}$ sequences contain more than $s/3$ faulty items is less than 2^{-n}.

Proof: By Lemma 2 the difference in the number of bits required to represent all sequences and the number of bits required to represent all the "bad" sequences is $\sqrt{m}\left(\frac{s\epsilon}{3} + lg\ n + 1 - lg\ m\right) - n$. For $m = 4(n+1)^2$ and $s = 9lg\ (n+1)/\epsilon + 2$ the difference is greater than n. Therefore, less than $1/2^n$ of all possible sequences have more than $\sqrt{m}$ subsequences with more than $s/3$ faulty items. □

Corollary 3 indicates that for m large enough any randomly chosen sequence of m sequences of s items will turn out to be "bad" with an arbitrarily small probability.

6. The existence of a consensus algorithm with O(lg n) expected number of rounds

Theorem 6. If $n/t > 3+\delta$ for some constant $\delta > 0$, then there is a consensus algorithm whose expected running time is $O(lg\ n)$.

Proof: By Lemma 2 there is a sequence of m sequences of s processes such that size $s = O(lg\ n)$, and less than $\sqrt{m}$ sequences have more than $s/3$ faulty processes. By creating a compound process from each sequence we can construct a set π of m compound processes such that less than $\sqrt{m}$ of them are *FAULTY*. Let $q_1, \ldots, q_k$ be the processes that are not included in any of the sets in π. The algorithm consists of just applying Algorithm 2 to $\pi \cup \{q_1, \ldots, q_k\}$ with $q_1, \ldots, q_k$ participating as spectator processes. By Theorem 5 and Corollary 1 the algorithm is a consensus algorithm for $n > 5$. By Theorem 4 the expected number of "virtual phases" to reach agreement is constant. Each "virtual phase" takes $5(s+1)/3 = O(lg\ n)$ actual rounds. Thus the expected running time of the algorithm is $O(lg\ n)$. □

The message complexity of the algorithm is as follows. In a round where processes *SEND* messages $O(n^4(lg\ n)^2)$ actual messages. In a round where processes do not *SEND* messages only $O(n^2(lg\ n)^2)$ actual messages are sent.

Note that the constants hidden under the O-notation are not necessarily large. For $\delta = 3$, $s = 18\ lg\ n + 1$, and by Corollary 2 the expected number of "virtual phases" can be reduced to 3 with a slight increase in s.

7. Discussion and conclusion

We showed that we can reduce the proportion of faulty processes in a system by aggregating them into compound processes. The reduction in the proportion of the faulty process enabled us to achieve consensus in expected $O(lg\ n)$ rounds. This reduction from a constant proportion to an $O(1/\sqrt{n})$ is significant and quite unintuitive; while it is reasonable to assume that the probability that a process is faulty is some small constant, it is not likely at all that this probability will decrease while the total number of processes increases (as in $O(1/\sqrt{n})$).

An interesting open problem is how to explicitly construct the set of m compound processes with the desired property. However, even without an explicit construction, Corollary 3 indicates that if m is large enough ($m > 4n^2$), then with an overwhelming probability (greater than $1-2^{-n}$), there will be less than $\sqrt{m}$ faulty processes in any randomly chosen construction. Other open problems are whether one can weaken the assumptions about the faulty processes' "knowledge of the system", and establishing a lower bound for the expected number of rounds in probabilistic Byzantine Generals algorithms.

Acknowledgement

I would like to thank Benny Chor, Oded Goldreich, and T. K. Srikanth for their useful comments.

Reference

BenO83 M. Ben-Or, Another advantage of free choice: completely asynchronous agreement protocols, *Proceeding 2nd ACM Symposium on Principles of Distributed Computing,* Montreal, Canada, pp. 27-30, August 1983.

Brac83 G. Bracha and S. Toueg, Resilient consensus protocols, *Proceeding 2nd ACM Symposium on Principles of Distributed Computing,* Montreal, Canada, pp. 12-26, August 1983.

Brac84a G. Bracha, An n/3 resilient consensus protocol, to appear in *Proceeding 3rd Symposium on Principles of Distributed Computing,* Vacouver, Canada, August 1984.

Brod83 A. Z. Broder and D. Dolev, Flipping coins in many pockets, manuscript. (IBM Research Center, San Jose, California.)

Chor84 B. Chor, and B. Coan, A simple and efficient randomized Byzantine agreement algorithm, *MIT/LCS/TM-266,* June 1984.

Dole81 D. Dolev, Unanimity in an unknown and unreliable environment, *Proceedings 22nd Annual Symposium on Foundations of Computer Science,* Nashville, Tennessee, pp. 159-168, Oct. 1981.

Dole82a D. Dolev, The Byzantine Generals strike again, *Journal of Algorithms,* vol. 3, no. 1, pp. 14-30, April 1983.

Dole82b D. Dolev, Polynomial algorithm for multiple process agreement, *Proceeding 14th Annual ACM Symposium on Theory of Computing,* San Francisco, California, pp. 404-407, May 1982.

Fisc82a M. J. Fischer and N. A. Lynch, A lower bound on the time to assure interactive consistency, *Information Processing Letters,* vol. 14, no. 4, pp. 183-186, May 1982.

Lamp82a L. Lamport and M. Fischer, Byzantine Generals and Transaction Commit protocols, Opus 62, SRI International, April 1982.

Lamp82b L. Lamport, R. Shostak and M. Pease, The Byzantine Generals problem, *ACM Transactions on Programming Languages and Systems,* vol. 4, no. 3, pp. 382-401, July 1982.

Rabi83 M. Rabin, Randomized Byzantine Generals, *Proceeding 24th Symposium on Foundations of Computer Science,* Tuscon, Arizona, pp. 403-409, Nov. 1983.

Rive78 R. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and Public-Key cryptosystems, *Communications of the ACM,* vol. 21, no.2, pp. 120-126, Feb. 1978.

Sham79 A. Shamir, How to share a secret, *Communications of the ACM,* vol. 22, no. 11, pp. 612-613, Nov. 1979.

Yao83 A. C. Yao, On the succession problem for Byzantine generals, Tech. Rep., Computer Science Dept., Stanford University, to appear.

Appendix

Quoted here are the cryptographic assumptions of [Brod84].

"A public key cryptosystem consists of two functions on a certain message space: an encryption function E that is public, and a decryption function $D = E^{-1}$ that is private. It is assumed that knowing only E and $E(x)$, it is not feasible to compute x in polynomial time with more than an exponentially small probability of success."

"For our purposes we need a family of cryptosystems that have certain extra properties:

1. A processor in the network can generate a cryptosystem in the family in polynomial time , and there is a rule that ensures that each process generates a different cryptosystem.

2. Every cryptosystem in the family is such that the encryption function is one to one.

3. Given an encryption function E produced by a certain processor, it is possible to check in polynomial time that: (a) the rule prescribed by property 1 was obeyed, and (b) the implied cryptosystem belongs to the family (hence E is one to one).

4. We assume that determining the value $x \; mod \; m$, for any m, from the knowledge of the encryption function E, of the value $E(x)$, and of the fact that $0 \leq x \leq p$, is as hard as breaking the system if p suitably large.

In practice such a family of cryptosystem can be based on the RSA [Rive78] scheme with a proper choice of parameters."