The Weighted Byzantine Agreement Problem

Vijay K. Garg and John Bridgman
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712-1084, USA
garg@ece.utexas.edu, johnfb@mail.utexas.edu

***Abstract*—This paper presents a weighted version of the Byzantine Agreement Problem and its solution under various conditions. In this version, each machine is assigned a weight depending on the application. Instead of assuming that at most f out of N machines fail, the algorithm assumes that the total weight of the machines that fail is at most f/N. When each machine has weight $1/N$, this problem reduces to the standard Byzantine Generals Agreement Problem. By choosing weights appropriately, the weighted Byzantine Agreement Problem can be applied to situations where a subset of processes are more trusted. By using weights, the system can reach consensus in the presence of Byzantine failures, even when more than $N/3$ processes fail, so long as the total weight of the failed processes is less than $1/3$. Also, a method to update the weights of the processes after execution of the weighted Byzantine Agreement is given. The update method guarantees that the weight of any correct process is never reduced and the weight of any faulty process, suspected by correct processes whose total weight is at least $1/4$, is reduced to 0 for future instances. A short discussion of some weight assignment strategies is also given.**

I. Introduction

The Byzantine Agreement (BA) [1], [2] is a fundamental problem in distributed computing with extensive literature [3]–[5]. In the usual set-up, there are N processes required to agree on a common value, given that at most f of them may show arbitrary or Byzantine behavior. In real-life applications, there may be multiple classes of processes. For example, in a mobile computing scenario, mobile hosts may be more likely to fail compared to mobile stations. In another example, processes in the same data center may be more likely to fail together. This paper defines a weighted version of the Byzantine Agreement Problem (WBA) and provides lower bounds and algorithms for it. In WBA, each process P_i is assigned a weight $w[i]$, such that $0 \leq w[i] \leq 1$ and the sum of all weights is 1. The WBA problem requires a protocol to reach consensus when the total weight of the failed processes is at most ρ. The weighted version gives some surprising results for the BA problem. First, even if greater than $N/3$ processes are Byzantine, the system can still reach consensus so long as ρ is less than $1/3$. This result is quite useful in the system with a small set of trusted processes and a large set of less trusted processes.

Secondly, the message complexity and the number of rounds required to achieve consensus for the weighted version is shown to always be less than or equal to those for the unweighted version. Suppose the system must tolerate $\rho = f/N$ for any integer f such that $0 \leq f < N/3$. It is known that any protocol for BA requires at least $f + 1$ rounds [6]. The unweighted version of the Queen algorithm [7] requires $f + 1$ rounds, each with two phases. In this paper, the notion of *anchor* of a system (denoted by α_ρ) is defined as the least number of processes whose total weight exceeds ρ. This paper shows that WBA can be solved with the number of rounds equal to the system's anchor. The anchor for a system with $\rho = f/N$ is $f + 1$ at most and, in many cases, much smaller than $f + 1$. Two algorithms for the WBA problem are given: the weighted-Queen algorithm and the weighted-King algorithm. These algorithms are generalizations of the algorithms proposed by Berman and Garay [7] and Berman, Garay and Perry [8]. The weighted-Queen algorithm takes α_ρ rounds, each with two phases, and can tolerate any combination of failures so long as $\rho < 1/4$. The weighted-King algorithm takes α_ρ rounds, each with three phases, and can tolerate any combination of failures so long as $\rho < 1/3$. The weighted version of BA gives a general framework to study many algorithms by instantiating BA with different weights. When the weight vector is $(1, 0, 0, 0..0)$, our algorithm reduces to a centralized algorithm, where the first process is expected not to fail and any number of other processes may fail. If M out of N processes are considered more trusted, two classes of processes can be specified by setting the weight of the M trusted processes to $1/M$ and 0 for rest. The traditional BA problem is represented by setting all the weights to $1/N$.

A general approach to the problem of knowing some kind of structure to the ways processes can fail, has been considered by Hirt and Maurer [9]. Hirt and Maurer come up with what is called an adversarial structure. This structure is the set of all subsets of processes whose failure must be tolerated. Consider a set of processes $P = \{d, e, f, g, h, i\}$; then, the adversarial structure A would be all subsets of P that should be tolerated. Hirt and Maurer show that as long as the union of any three sets in the adversarial structure does not contain all processes in P; then, an algorithm for Byzantine agreement that can tolerate that adversarial structure exists. The basis of A, which is all maximal sets in A, is $\bar{A}$.

$$\bar{A} = \{\{d, e, f\}, \{d, g\}, \{e, h\}, \{e, i\}, \{f, g\}\}$$

The traditional BA algorithm can only handle one faulty process from P; but, there exists an algorithm that can handle at least one additional faulty process for any one faulty process as indicated by $\bar{A}$. Fitzi and Maurer [10] give an algorithm that can perform Byzantine agreement on such an adversarial

structure. The algorithm by Fitzi and Maurer has message complexity polynomial in the number of processes and round complexity of less than two times the number of processes. The round complexity for our algorithms are optimal. For this particular example, a weight assignment can be found so that the King algorithm presented here can tolerate all processes in one of the subsets of P in A being faulty. One such weight assignment is shown in Table I. Also, the adversarial structure may be exponentially larger than the weight assignment. The algorithms presented in this paper are considerably simpler than the one presented by Fitzi and Maurer.

TABLE I
WEIGHT ASSIGNMENTS FOR P WHICH SATISFY THE ADVERSARIAL STRUCTURE A.

Process	d	e	f	g	h	i
Weight	$1/9$	$1/18$	$8/57$	$1/16$	$5/19$	$5/19$

Others have considered the use of artificial neural networks (ANN) for BA [11], [12]. The ideas in this paper could also be extended to use ANN. Randomization or authentication is not assumed to be available. There are many algorithms for BA with randomization [13], [14] or with authentication [1], [15]. The method of using weights and updating weights presented here is expected to be applicable in these settings as well.

Methods to update weights for future rounds of WBA are also discussed. The weight update method guarantees that the weight of a correct process is never reduced and the weight of any faulty process, suspected by correct processes whose total weight is at least $1/4$, is reduced to 0. Initial weight assignment is application specific. Some ideas for weight assignment and resulting probabilities and round complexity are discussed.

The organization of the rest of this paper is as follows. First, the model of the system that runs algorithms is defined. Second, the Weighted-Queen Algorithm is given. Then, the Weighted-King Algorithm is discussed. Next, a simple weight update method is given. Then, some initial weight assignment strategies are presented. Finally, concluding remarks are made.

II. SYSTEM MODEL

The system model used in this paper is a distributed system with N processes, $P_1..P_N$, with a completely connected topology. The underlying system is assumed to be *synchronous*; i.e., there is an upper bound on the message delay and on the duration of actions performed by processes. The model assumes that processes may fail; but, the underlying communication system is reliable and satisfies first-in first-out (FIFO) message ordering. The processes may fail in an arbitrary fashion; in particular, they may lie and collude with other failed processes to foil any protocol. The processes that do not fail in any computation are called *correct* processes. Assume that there is a non-negative weight $w[i]$ associated with each process P_i. All processes in the system have complete knowledge of weights of all the processes. For simplicity, assume that weights are normalized; i.e., the sum of all weights is one. Let ρ be the sum of weights of all failed processes. This paper assumes that ρ is strictly less than one.

The Weighted Byzantine Agreement (WBA) problem can be specified as follows. All processes propose a binary value with the goal of deciding on one common value. Given a weight assignment to all processes, and the assumption that the weight of processes that fail during the execution is at most ρ, the WBA problem is to design a protocol that satisfies the following conditions:

- **Agreement**: Two correct processes cannot decide on different values.
- **Validity**: The value decided must be proposed by some correct process.
- **Termination**: All correct processes decide in finite number of steps.

The following lower bounds follow easily from the standard BA lower bound arguments.

Lemma 1: There is no protocol to solve the WBA problem for all values of w when $\rho \geq 1/3$.

Proof: Any protocol to solve WBA can be used to solve standard BA by setting $w[i] = 1/N$ for all i. For this weight assignment, $\rho \geq 1/3$ implies that the number of failed processes f in the standard BA protocol is at least $N/3$. It is well-known that no protocol exists for standard BA when $3f \geq N$ [1]. ■

For simplicity, also assume that weights associated with P_i are in non-increasing order. This can be achieved by renumbering processes, if necessary. Given any ρ and weight assignment w, define the *anchor* α_ρ as the minimum number of processes such that the sum of their weights is strictly greater than ρ. Formally,

$$\alpha_\rho = min\ \{k \mid \sum_{i=1}^{i=k} w[i] > \rho\}.$$

To get an insight into α_ρ, consider the case when ρ is f/N and each process has equal weight $1/N$. In this case, α_ρ equals $f + 1$. The significance of α_ρ is that at least one process from $P_1..P_{\alpha_\rho}$ is guaranteed to be correct. When ρ is zero, α_ρ is 1. The largest possible value of α_ρ is N, because $\rho < 1$ by assumption. The following lower bound on the number of rounds for any consensus protocol is obtained from standard consensus arguments.

Lemma 2: Any protocol to solve the WBA problem for a system with $\rho < 1$ takes at least α_ρ rounds of messages, in the worst case.

Proof: If not, a protocol exists to solve BA in less than $f + 1$ rounds when all weights are uniform. ■

III. WEIGHTED-QUEEN ALGORITHM

In this section, an algorithm is given that takes α_ρ rounds, each round of two phases, to solve the WBA problem. The algorithm is based on the unweighted version of the algorithm given by Berman and Garay [7]. The algorithm uses constant-size messages, but requires that $\rho < 1/4$. Each process has a preference for each round, which is initially its input value. The algorithm shown in Fig. 1 is based on the idea of a rotating queen (or coordinator). Processor P_i is assumed to

```
P_i::
var
 V: {0,1}  initially proposed value;
 w: const array[1..N] of weights;
      initially ∀j : w[j] ≥ 0 ∧ (∑_j : w[j] = 1)

 for q := 1 to α_ρ do

   float s0,s1 := 0.0, 0.0;

   first phase :
     if (w[i] > 0) then
       send V to all processes including itself;
     forall j such that w[j] > 0 do
       if 1 received from P_j then
         s1 := s1 + w[j];
       else if
       0 received from P_j or no message from P_j
       then
         s0 := s0 + w[j];
     if (s1 > 1/2) then
       myvalue := 1; myweight := s1;
     else myvalue := 0; myweight := s0;

   second phase:
     if (q = i) then
       send myvalue to all other processes;
     receive queenvalue from P_q;
     if myweight > 3/4 then
       V := myvalue;
     else V := queenvalue;

 endfor;

 output V as the decided value;
```

Fig. 1. Queen algorithm for Weighted Byzantine Agreement at P_i

be the queen for round i. In the first phase of a round, each process exchanges its value with all other processes. Based on the values received and the weights of the processes sending these values, the process determines its estimate in the variable $myvalue$. In the second phase, the process receives the value from the queen. If P_i receives no value (because the queen has failed), then P_i assumes 0 (a default value) for the queen value. Now, P_i decides whether to use its own value or the $queenvalue$. This decision is based on the sum of the weights of the processes which proposed $myvalue$ given by the variable $myweight$. If $myweight$ is greater than $3/4$, then $myvalue$ is chosen for V; otherwise, $queenvalue$ is used. The correctness of the protocol is shown by the following sequence of lemmas.

Lemma 3 (Persistence of Agreement): Assuming $\rho < 1/4$, if all correct processes prefer a value v at the beginning of a round; then, they continue to do so at the end of the round.

Proof: If all correct processes prefer v, then the value of $myweight$ for all correct processes will at least be $3/4$; because, ρ is at most $1/4$. Hence, they will choose $myvalue$ in the second phase and ignore the value sent by the queen. ■

Lemma 4: There is at least one round in which the queen is correct.

Proof: By assumption, the total weight of processes that have failed is ρ. The for loop is executed α_ρ times. By definition of α_ρ, there exists at least one round in which the queen is correct. ■

Now the correctness of the protocol can be shown.

Theorem 1: The algorithm in Fig. 1 solves the agreement problem for all $\rho < 1/4$.

Proof: The validity property follows from the persistence of agreement. If all processes start with the same value v, then, v is the value decided. Termination is obvious because the algorithm takes a fixed number of rounds. Next, the agreement property is shown. From Lemma 4, at least one of the rounds has a correct queen. Each correct process decides either on the value sent by the queen in that round or its own value. It chooses its own value w only if $myweight$ is at least $3/4$. Therefore, the queen of that round must have weight of at least $1/2$ for that value; because, at most $1/4$ of the weight in P_i is from faulty processes. Thus, the value chosen by the queen is also w. Hence, each process decides on the same value at the end of a round in which the queen is non-faulty. From persistence of agreement, the agreement property at the end of the algorithm follows. ■

Let us analyze the algorithm's message complexity. There are α_ρ rounds, each with two phases. In the first phase, all processes with positive weight send messages to all processes. This phase results in pN messages where $p \leq N$ is the number of processes with positive weight. The second phase uses only N messages. Thus, the total number of messages is $\alpha_\rho(pN+N)$. The number of messages can be further reduced by sending messages to zero weight processes only in the last round. Note that the algorithm from [7] takes $f+1$ rounds (each with two phases) when the maximum number of allowed failures is f. The following lemma shows that the number of rounds for the weighted version is at most the number required for the unweighted version.

Lemma 5: $\alpha_{f/N} \leq f+1$ for all w and f.

Proof: It is sufficient to show that for all f, $\sum_{i=1}^{i=f} w[i] \geq f/N$. Suppose $\sum_{i=1}^{i=f} w[i] < f/N$ for some f. This implies that the sum of the remaining weights is $\sum_{i=f+1}^{i=n} w[i] > (N-f)/N$, because all weights add up to 1. Since w is in nondecreasing order, $w[f+1] > 1/N$; otherwise, the sum of the remaining weights would be at most $(N-f)/N$. But, this implies that $\sum_{i=1}^{i=f} w[i] > f/N$, because $w[i]$ for all $i \leq f$ is at least $w[f+1]$. This contradicts our original assumption. ■

IV. Weighted-King Algorithm

This section gives an algorithm that takes α_ρ rounds with three phases per round to solve the WBA problem. The algo-

rithm is based on the *Phase King* algorithm by Berman, Garay and Perry [8]. The King algorithm only requires $\rho < 1/3$; but, adds an additional phase per round compared to the Queen algorithm. The King algorithm is given in Fig. 2. As in the Queen algorithm, the King algorithm has a rotating coordinator. It is assumed that the coordinator for round k is process P_k. Each process P_i has a current preference V which can be 0, 1, or $undecided$. Initially, for every P_i, V is either 0 or 1.

In the first phase, if process P_i has a positive weight, then P_i sends V to all processes including itself. Next, P_i receives values from every process with positive weight and adds up the cumulative weight of processes that propose 0 as $s0$ and 1 as $s1$. If $s0$ or $s1$ are greater than or equal to $2/3$, then P_i sets its preference V to the corresponding value; otherwise, P_i sets its preference to $undecided$.

In phase two, P_i first sends its new preference of V to every process if P_i's weight is positive and resets $s0$, $s1$, and su. Note that in this phase, unlike in phase one, processes may propose the value $undecided$. Then, P_i receives from all processes with positive weights and accumulates the sum of weights of processes into $s0$ for processes who propose 0, into $s1$ for processes who propose 1 and into su for processes who propose $undecided$. The final step in phase two is for P_i to set its preference to a new value based on the cumulative weights computed in the first part of this phase. If one of the cumulative weights is greater than $1/3$, P_i sets its preference to that value. If more than one of the sums is greater than $1/3$, P_i gives preference to 0, then 1, then $undecided$. P_i also sets $myweight$ to the cumulative weight of the value that V is set.

In phase three, if P_i is the king for the current phase, P_i sends its preference V to every process. Next, all processes receive the king's value into $kingvalue$. Then, if P_i is undecided ($V = undecided$) or the weight stored in $myweight$ from phase two is less than $2/3$, P_i sets its preference to $kingvalue$ if $kingvalue$ is not $undecided$ or 1 if $kingvalue$ is $undecided$. After executing for α_ρ rounds, P_i outputs V as the decided value. The correctness of the King algorithm is shown in the following lemmas.

Lemma 6 (Persistence of Agreement): Assuming $\rho < 1/3$, if all correct processes prefer a value v at the beginning of a round; then, they continue to do so at the end of the round.

Proof: If all correct processes agree at the beginning of the round; then, for the first phase, by definition, the same value must be chosen as $\rho < 1/3$. For the second phase, the same value must again be chosen as $\rho < 1/3$. For the third phase, because all correct processes agree and $\rho < 1/3$, all correct processes will ignore the king's value and keep their own. ■

Lemma 7: There is at least one round in which the king is correct.

Proof: By assumption, the total weight of processes that have failed is less than ρ. The for loop is executed α_ρ times. By definition of α_ρ, there exists at least one round in which the king is correct. ■

Theorem 2: The algorithm in Fig. 2 solves the agreement

```
P_i::
var
 V: {0, 1, undecided}  initially proposed value;
 w: const array[1..N] of weights;
      initially ∀j : w[j] ≥ 0 ∧ (∑_j : w[j] = 1)

 for k := 1 to α_ρ do

   float s0,s1,su := 0.0, 0.0, 0.0;

   first phase :
     if (w[i] > 0) then
        send V to all processes including itself;
     forall j such that w[j] > 0 do
        if 1 received from P_j then
           s1 := s1 + w[j];
        else if 0 received from P_j then
           s0 := s0 + w[j];
     if (s0 ≥ 2/3) then V := 0;
     else if (s1 ≥ 2/3) then V := 1;
     else V = undecided;

   second phase:
     s0,s1,su := 0.0, 0.0, 0.0;
     if (w[i] > 0) then
        send V to all processes including itself;
     forall j such that w[j] > 0 do
        if 1 received from P_j then
           s1 := s1 + w[j];
        else if 0 received from P_j then
           s0 := s0 + w[j];
        else su := su + w[j];
     if (s0 > 1/3) then
        V := 0; myweight := s0;
     else if (s1 > 1/3) then
        V := 1; myweight := s1;
     else if (su > 1/3) then
        V := undecided; myweight := su;

   third phase:
     if (k = i) then send V to all other processes;
     receive kingvalue from P_k;
     if V = undecided or myweight < 2/3 then
        if kingvalue = undecided then V = 1
        else V = kingvalue

 endfor;

 output V as the decided value;
```

Fig. 2. King algorithm for Weighted Byzantine Agreement at P_i

problem for $\rho < 1/3$.

Proof: Validity is satisfied by persistence of agreement. If all processes start with the same value, then that value will be decided. Termination is obvious because the algorithm takes a fixed number of rounds. From Lemma 7, in at least one round, the king will be correct. In that round, every correct process will choose either the king's value, 1, or its own value. The only way that a process may choose its own value is if $myweight \geq 2/3$ and the process is not undecided; otherwise, the process will choose the king's value or 1 if the king is undecided. If a process chooses its own value, then, $myweight \geq 2/3$ for that process and the weight of its value V will be $\geq 1/3$. So, the king must also have chosen the same value. If $myweight < 2/3$ or V is undecided, then the process will choose the kings value or 1 if the king is undecided. Because the king is correct, then all processes will choose the same value. ∎

The King algorithm takes α_ρ rounds with three phases per round. In phase one and two, each process with positive weight sends N messages. In phase three, the king process sends N messages. This results in $\alpha_\rho(2pN + N)$ messages where p is the number of processes with positive weight.

V. Updating Weights

In this section, the case when the system is required to solve BA multiple times is considered. This case arises in most real-life applications of BA, such as, maintenance of replicated data and fault-tolerant file systems [16]. In addition, each execution of the BA protocol provides certain feedback in terms of the processes' behavior. For example, if a process did not follow the protocol (i.e., did not send the required messages), it should be considered less reliable for future BA instances. In this section, a fault-tolerant method to update weights is given. For simplicity, only the weighted-Queen algorithm is given; the extension to weighted-King algorithm is similar.

The following lemma gives the conditions sufficient for P_i to detect that P_j is faulty.

Lemma 8: In the Weighted-Queen algorithm, a correct process P_i can detect that P_j is faulty if any of the following conditions are met:

1) If P_j either does not send a message or sends a message with wrong format in any of the rounds, then P_j is faulty.
2) If $myweight > 3/4$ in any round and the value sent by the queen in that round is different from $myvalue$, then the queen is faulty.

Proof: The first part is obvious. For the second part, note that if $myweight > 3/4$; then, the weight for the queen for that value in that round is at least $1/2$. If the queen were correct, the value sent by the queen would have matched $myvalue$. ∎

The algorithm in Fig. 1 is modified by adding a variable $faultySet$ that keeps track of all processes that P_i has detected to be faulty based on Lemma 8. Now a method is presented to update the weights of the processes such that with every execution of WBA, the processes get better in solving WBA by increasing the weights of reliable processes. These algorithms require that the weight assignment for all correct processes be identical; so, it is not sufficient for a process to update its weight individually. All correct processes need to agree on the faulty set.

The algorithm to update weights shown in Fig. 3 consists of three phases. In the first phase, called the learning phase, processes broadcast their $faultySet$ to learn about faulty processes from other correct processes. The main idea is that if processes with total weight at least $1/4$ inform P_i that some process P_j is faulty, then P_j is in $faultySet$ of at least one correct process. The second phase consists of processes agreeing on the set of faulty processes. For each process j, if j is in the $faultySet$ of P_i , then P_i invokes `Weighted-Queen-BA` algorithm with 1 as the proposed value; otherwise, it invokes it with 0 as the proposed value. The output variable $value$ denotes the decided value by the `Weighted-Queen-BA` algorithm. Therefore, the set of faulty processes that all correct processes agree upon is $consensusFaulty$. In the third phase, processes update their weights based on $consensusFaulty$.

The correctness of the algorithm in Fig. 3 is shown in the following lemma and theorem.

Lemma 9: All correct processes with positive weights before the execution of the algorithm have identical w vectors after the execution of the algorithm.

Proof: The weight assignment is done based on $consensusFaulty$. The variable $consensusFaulty$ is identical at all correct processes based on the correctness of Weighted-Queen algorithm. ∎

Theorem 3: A correct process can never be in $consensusFaulty$. Any faulty process that is in the initial $faultySet$ of correct processes with total weight at least $1/4$ will be in $consensusFaulty$ of all correct processes.

Proof: A correct process P_j can never be in the initial $faultySet$ of any correct process (due to Lemma 8). In the learning phase, $suspectWeight[j]$ at any process can never be equal or more than $1/4$, because only faulty processes can suspect P_j. Therefore, j is not in $faultySet$ of any correct process after the learning phase. Since all correct processes will invoke WBA with 0 for P_j, by validity of the Weighted-Queen-BA algorithm, it will not be in $consensusFaulty$. Any faulty process that is in the initial $faultySet$ of correct processes with total weight of at least $1/4$ will be in $faultySet$ of all correct processes after the learning phase. Again, from the validity of WBA, the faulty process will be in $consensusFaulty$. ∎

The model assumed here for updating weights is that once a process is faulty, it will always be faulty. A modification can be considered where a process may become non-faulty after being faulty for a period of time. In this case, instead of setting the weight to zero, the weight can be reduced by some multiple.

```
P_i::
// Only processes with positive weights participate
// in this algorithm
var
  faultySet: set of processes based on Lemma 8;
  consensusfaulty: set of processes initially {};
  suspectWeight: array[1..p] of float initially all 0.0;

  First phase (learning phase):
    forall j do
      send faultySet to all (including itself);
    forall j do
      receive faultySet_j from P_j;
      forall k ∈ faultySet_j do
        suspectWeight[k] :=
          suspectWeight[k] + w[j];
    forall j do
      if suspectWeight[j] ≥ 1/4 then
        faultySet := faultySet ∪ {j};

  Second phase:
    // Do WBA on each of the processes to see
    // if they are faulty
    forall j do
      if j ∈ faultySet then
        value := Weighted-Queen-BA(1);
      else value := Weighted-Queen-BA(0)
      if (value = 1) then
        consensusFaulty := consensusFaulty ∪ {j};

  Third phase:
    // set weight of faulty processes to 0
    float totalWeight := 1.0;
    forall j ∈ consensusFaulty do
      totalWeight := totalWeight − w[j];
      w[j] := 0;
    // renormalise weights
    forall j do
      w[j] := w[j]/totalWeight;
```

Fig. 3. Weight-Update Algorithm for the Queen algorithm for Weighted Byzantine Agreement at P_i

VI. Weight Assignment

Deciding what weight assignment to use is application specific. A simplified example will be considered for this section. Consider two sets of processes A and B where all processes in A have probability of failure f_a and all processes in B have probability of failure f_b. We will consider four weight assignments. The first is a uniform weight assignment for everyone. This weight assignment produces the same results as the classical algorithm. The next assignment is to only give non-zero weights to the set with a lower probability of failure. The third is to give weights to each process proportional to the inverse of their probability of failure. Weights proportional to the probability of not failing is the final assignment considered.

The graph in Fig. 4 is the probability of the weight of failed processes exceeding $1/3$ versus $|B|$ with $|A| = 6, f_a = 0.1, f_b = 0.3$. This graph is only taken for points where the number of processes is divisible by three. The number three is chosen because taking every point produces many more jumps in the graph which just add noise and distract from the trend. Notice that there are still some jumps. These jumps are caused by the effect of adding a process to a group where that addition does not increase the number of faulty processes that can be tolerated. But, adding that one process increases the expected weight of failed processes. So, there is a jump in the probability of the total weight of failed processes being above $1/3$. Each curve starts at the same value as set B is empty. Observe that each curve initially has a positive average slope. It is not until a significant number of additional processes are added that the curve begins to have a negative slope. The uniform assignment gives the worse probability for a small size of B relative to the size of A. Changing the number of processes in set A moves the curves vertically in relation to each other. Which weight assignment is best depends upon the number of processes in both A and B and their probability of failure. In this particular example, setting the weight proportional to the inverse probability of failure gives the best results.

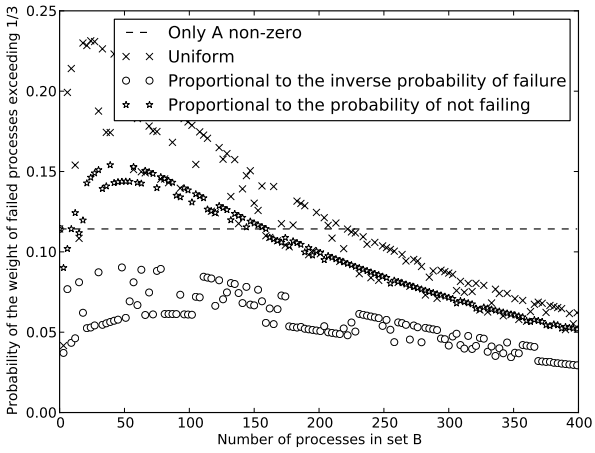

Fig. 4. Probability of the weight of failed processes exceeding 1/3 with versus $|B|$ with $|A| = 6, f_a = 0.1, f_b = 0.3$.

Fig. 5 shows the number of rounds required for the King algorithm to ensure success. Notice the uniform assignment is the highest. In both of these graphs, the uniform weight assignment was the least attractive. The most attractive assignments are only giving positive weights to group A and setting the weight proportional to the inverse probability of failure. For this particular setup, setting weights proportional to the inverse probability of failure is the best. When the size of set B is not much larger than A, only giving positive weights to set A may be the best. When the size of set B is significantly

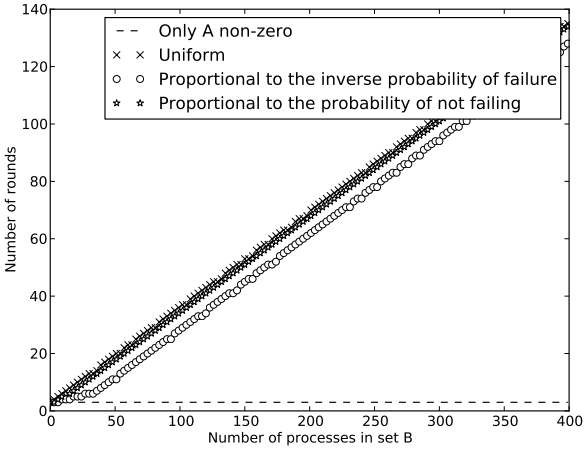

Fig. 5. The number of rounds required for the King algorithm versus the number of processes in set B for the different weight assignments.

larger than A, then setting the weights to be proportional to the inverse probability of failure is the best.

VII. Conclusions

This paper has presented a weighted version of the Byzantine Agreement Problem and provided solutions for the problem in a synchronous distributed system. We show that the weighted version has the advantage of using fewer messages and tolerating more failures (under certain conditions) than is required by the lower bound for the unweighted version. These algorithms have applications in many systems in which there are two classes of processes: trusted and untrusted processes. Instead of tolerating any f faults in the BA problem, these algorithms tolerate failure of processes with total weight less than f/N. For example, an implementation can now tolerate more than f faults of untrusted processes; but, fewer than f faults of trusted processes depending on the weight assignment. A fault-tolerant method has also been presented to update the weights at all the correct processes. This algorithm is useful for many applications where the agreement is required multiple times. Our update algorithm guarantees that the weight of a correct process is never reduced and the weight of any faulty process, suspected by correct processes whose total weight is at least $1/4$, is reduced to 0.

References

[1] M. Pease, R. Shostak, and L. Lamport, "Reaching agreements in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980.

[2] L. Lamport, R. E. Shostak, and M. C. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[3] P. Feldman and S. Micali, "An optimal probabilistic protocol for synchronous byzantine agreement," *SIAM J. Comput.*, vol. 26, no. 4, pp. 873–933, 1997.

[4] D. Dolev, R. Reischuk, and H. R. Strong, "Early stopping in byzantine agreement," *J. ACM*, vol. 37, no. 4, pp. 720–741, 1990.

[5] J. A. Garay and Y. Moses, "Fully polynomial byzantine agreement in t + 1 rounds," in *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1993, pp. 31–41.

[6] D. Dolev and H. R. Strong, "Polynomial algorithms for multiple processor agreement," in *Proceedings of the ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1982, pp. 401–407.

[7] P. Berman and J. A. Garay, "Asymptotically optimal distributed consensus," in *ICALP: Proceedings of the International Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, 1989, pp. 80–94.

[8] P. Berman, J. Garay, and K. Perry, "Towards optimal distributed consensus," in *Foundations of Computer Science*, 30 1989, pp. 410 –415.

[9] M. Hirt and U. Maurer, "Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract)," in *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1997, pp. 25–34.

[10] M. Fitzi and U. M. Maurer, "Efficient byzantine agreement secure against general adversaries," in *DISC '98: Proceedings of the 12th International Symposium on Distributed Computing*. London, UK: Springer-Verlag, 1998, pp. 134–148.

[11] S. C. Wang and S. H. Kao, "A new approach for byzantine agreement," in *Proceedings of the The International Conference on Information Networking*. Washington, DC, USA: IEEE Computer Society, 2001, p. 518.

[12] K.-W. Lee and H.-T. Ewe, "Performance study of byzantine agreement protocol with artificial neural network," *Inf. Sci.*, vol. 177, no. 21, pp. 4785–4798, 2007.

[13] M. O. Rabin, "Randomized byzantine generals," in *Foundations of Computer Science*. IEEE, 1983, pp. 403–409.

[14] G. Bracha, "An $O(\log n)$ expected rounds randomized Byzantine generals protocol," *Journal of the ACM*, vol. 34, no. 4, pp. 910–920, Oct. 1987.

[15] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM J. Comput.*, vol. 12, no. 4, pp. 656–666, 1983.

[16] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI*, 1999, pp. 173–186.