Optimistic agreement in asynchronous distributed systems

J. Roger Mitchell   Vijay K. Garg

TR   TR-PDS-1997-002   February   1997

Parallel & Distributed Systems group

Department of Electrical & Computer Engineering

University of Texas at Austin

Austin, Texas 78712

Optimistic agreement in asynchronous distributed systems

J. Roger Mitchell* Vijay K. Garg†

Parallel and Distributed Systems Laboratory
email: pdslab@ece.utexas.edu
Electrical and Computer Engineering Department
The University of Texas at Austin,
Austin, TX 78712

April 3, 1997

Abstract

Coordination among a group of processes is often performed in distributed systems. When failures can occur, this agreement on coordination becomes more difficult. Often, the agreement can take several rounds of message exchanges among the group. When a failure occurs, the latency increases, for example in a system where the group waits for a member to recover. In this paper we present a method for reducing the latency in these situations using optimism. We present a method for general agreement problems in which the agreement type is specified using two functions. We demonstrate how to do this for the problems of binary agreement, approximate agreement, and leader election. We give experimental results that show that this approach allows processes to perform useful work when they otherwise would have waited. For example, our implementation of approximate agreement allows processes to perform useful work for more than 80% of the period they would have otherwise waited (from 5 to 11 rounds).

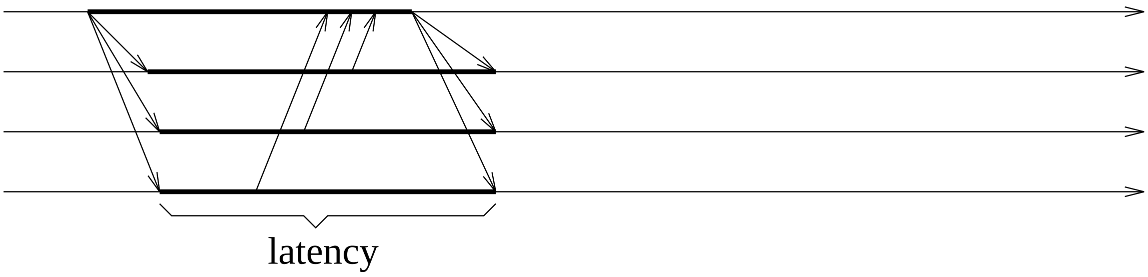
1 Introduction

In a distributed system it is often necessary to coordinate some action among a group of processes. The tasks of synchronizing clocks, agreeing on replicated sensor values, electing a leader, and agreeing on which processes are in a group are just a few examples. In the real world the environment in which these tasks are performed is faulty. The combination of this environment and these tasks constitute the agreement problem. Coordination in the agreement problem can be costly. This is seen in the number of messages required and the delay involved in reaching agreement, which typically requires all processes involved in the agreement to exchange messages in successive rounds. Here we focus on minimizing the impact of the delay. Figure 1 shows the delay associated with the simplest agreement protocol under error-free and failure in a distributed computation. During this

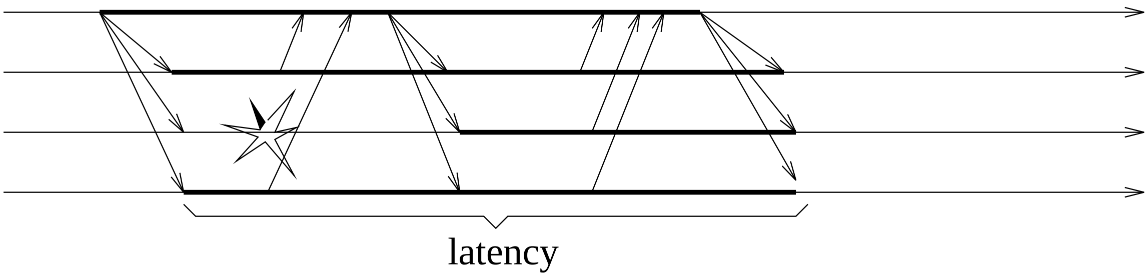
*supported in part by a Virginia and Ernest Cockrell fellowship

†supported in part by the NSF Grants ECS-9414780, CCR-9520540, a TRW faculty assistantship award, a General Motors Fellowship, and an IBM grant

time, a process must wait until it receives an agreed upon value which it can use in its computation. We examine a method for allowing a process to proceed with an assumed value for the agreement as soon as it needs it. Only if this assumption turns out to be wrong must it forfeit this optimistic execution. We apply the approach to the problems of binary agreement, approximate agreement, and leader election.



a) Latency under normal operation.



b) Latency during a process crash.

Figure 1: Agreement latency under error-free and faulty conditions.

1.1 Previous work

Prior work in solving the agreement problem in an asynchronous environment has included novel approaches to circumventing the impossibility result presented by Fischer, Lynch, and Paterson in [9]. This result states that there is no guarantee of agreement among a group of asynchronous processes in which even one failure can occur. [2, 3, 4, 8, 9, 6] give methods for solving agreement by using probabilistic methods, introducing synchrony, or weakening the problem. All these examples use rounds to reach agreement. The work of [4] uses failure detectors that suspect which processes have failed. [6] only reaches agreement of a value to within some predetermined difference.

Some previous work in database consistency, which is similar to agreement has also discussed using optimism. In [5] Davidson, et. al. discuss how a distributed database maintains consistency in a partitioned network. An optimistic scheme is discussed in which portions of the database may actually become inconsistent on opposite sides of a partition. Upon remerging, at least one side of the partition must undo previous work. In [1] Aggrawal and Bernstein give a method for employing optimism on distributed database reads. In this case, the quorum approach for database consistency is employed, and the first response is used optimistically. What we present is a general approach for any type of agreement to employ optimism.

Our work is presented beginning with necessary notation and functions. We then present the algorithm followed by its application to three agreement problems. Finally we analyze the probability that optimism will be worthwhile for these problems and give experimental results.

1.2 Notation

We use different failure assumptions for each agreement problem to which we apply optimism. However, in all the agreement problems we consider, we do not allow malicious process or channel behavior. That is, when processes fail they cease all activity and channels do not spuriously create or alter messages.

Our optimistic algorithm is designed to augment an agreement algorithm and sits between this algorithm and the application program. For our discussion we call any agreement algorithm a *base algorithm*. The optimistic algorithm is individualized for each agreement problem to which it is applied. The algorithm employs a set of data items called the *expected agreement set*, denoted by $\mathcal{S}$. This set is in addition to any data structures employed by the base algorithm. $\mathcal{S}$ can be thought of as a set containing all information about a currently active agreement that a particular process knows about. In most cases this set will simply contain proposed values. During an agreement the optimistic algorithm adds $\mathcal{S}$ to every message sent. In addition, $\mathcal{S}$ is also removed from every received message and combined with the local $\mathcal{S}$ using a union operation. Because of this union operation, it is apparent that $\mathcal{S}$ is a monotonically increasing set.

By including $\mathcal{S}$ in all messages during agreement, the information is spread as much as possible. Whenever a message reaches a process, the receiving process has the sum of information from its causal past. This method of piggybacking information is often called *gossipping* [11].

When a process executing the optimistic algorithm determines that the agreement has completed, $\mathcal{S}$ is assigned to $\mathcal{C}$, the *committed agreement set*. Although continuing to share $\mathcal{S}$ might allow other uncommited processes to commit sooner, to keep the algorithm simple, $\mathcal{S}$ is also cleared at this time. This will not affect the ability to commit at other processes, since the base algorithm is present. That is, the base algorithm must be able to commit independently. As long as this is possible, commitment will also occur with optimism.

To execute optimistically, the algorithm employs two functions $opt()$ and $com()$ which operate on the set $\mathcal{S}$. These functions are defined for the particular agreement problem being addressed. $opt()$ is the optimistic function which is designed to return a value if $\mathcal{S} \neq \{\}$; otherwise it returns null. $com()$ is the committable agreement function and will return a value only if that value is committable. Next we give the optimistic algorithm which employs $\mathcal{S}$, $opt()$, and $com()$.

2 The optimistic algorithm for agreement

2.1 Properties of $opt()$ and $com()$

Here we specify the properties that a function should possess to be used as $opt()$ or $com()$. We note that by their nature, $com()$ is a stricter function than $opt()$. That is, $com()$ should require

Process$_i$:

When making a proposal:
$\mathcal{S}$ = proposal for Process$_i$;
if ($opt(\mathcal{S}) \neq$ NULL) then
mark all results which depend on $opt(\mathcal{S})$ as optimistic;
begin optimistic execution.
(Optimistic execution restricts sends to other processes
participating in optimism).

Upon send of any message m:
attach $\mathcal{S}$ to m;

Upon receive of any message m:
remove $\mathcal{S}_m$ from m;
if ($opt(\mathcal{S}) \neq opt(\mathcal{S} \cup \mathcal{S}_m)$) then
remove all results marked optimistic;
$\mathcal{S} = \mathcal{S} \cup \mathcal{S}_m$;
if ($com(\mathcal{S}) \neq$ NULL) then
commit all results marked optimistic;
$\mathcal{S} = \{\}$;
else execute optimistically with $opt(\mathcal{S})$;

Figure 2: The optimistic algorithm

more strict conditions be met to return a value. If $opt()$ is as strict as $com()$ then optimism will not be possible.

Property 2.1 Maximum optimism: *$opt()$ should return non NULL when $\mathcal{S} \neq \{\}$.*

This property ensures that the algorithm will execute optimistically as early as possible. We note that other possible definitions include: $\mid \mathcal{S} \mid \geq 2 \Rightarrow opt(\mathcal{S}) \neq$ NULL.

Property 2.2 Valid optimism: *When $com(\mathcal{S}) \neq$ NULL, $opt(\mathcal{S}) = com(\mathcal{S})$.*

Property 2.3 Safety: *$com(\mathcal{S}) \neq$ NULL at P_i only if the base algorithm would commit to a value at P_i using the set $\mathcal{S}$.*

Property 2.4 Validity: *If the base algorithm commits to a value v at P_i then $com(\mathcal{S}) = v \pm \sigma$ at P_i, where σ is an allowable variance. (We add σ for use with approximate agreement algorithms. For all others $\sigma = 0$.)*

2.2 The algorithm

Figure 2 gives the optimistic algorithm which is designed to augment an agreement algorithm. As the figure indicates, the application program uses the return value of $opt(\mathcal{S})$ as if it were the decision value of the agreement. However, any variable that is changed because of *opt*() while *opt*() $\neq$ NULL and *com*() = NULL is marked. When this occurs the old value of the variable is saved. If *opt*() later changes, the marked variables can be restored and unmarked. This approach can be very quick and in general an application will often have some set of variables dependent on successive agreements. These variables can be saved as a single data structure at the beginning of optimism and quickly restored as necessary. In our implementation of the optimistic algorithm, we found that copying variables to a holding structure (1 Kbytes) was very small compared to the time required for an exchange of messages between a group of processes. The send/ack took 185 times longer than the copy[1]. Restoring variables was negligible, since we used a single pointer reassignment.

The functions *opt*() and *com*() are used to indicate whether optimistic or committable execution is possible. Note that when *com*() equals NULL but *opt*() does not, optimistic execution is possible. When *com*() does not equal NULL, committable execution is possible, and optimistic execution should no longer proceed. Note also that these functions define the beginning and end of agreement. When *opt*() changes from NULL to not NULL, agreement begins. When *com*() changes from NULL to not NULL, agreement ends.

3 When the algorithm is useful

Optimism is most useful in agreement problems where proposals are often commitable values. Sensor agreement, clock synchronization, and leader election are examples which have this property. The longer the optimism is required, the more useful the algorithm. This is directly connected to the latency of agreement, that is, the time to agree. Since agreement often requires multiple rounds, it is during this time that optimism can be employed.

4 Applications

In this section we present three applications of optimistic agreement: binary agreement, approximate agreement, and leader election.

4.1 Binary agreement

We present the use of optimism with a simple binary agreement (choosing between 0 and 1) under limited failures of an asynchronous system. To guarantee agreement (and to keep the algorithm simple), we assume that processes appear to never crash. That is, they may crash, but will eventually recover. Messages are also similarly reliable. The base algorithm in binary agreement then requires one round in which all processes send their own proposal and then receive proposals from all other processes. Once a proposal has been received from every other process, the base algorithm can commit, choosing the value that has a majority. In the case of no majority, the value 1 is chosen. The latency involved is the time it takes to send and collect proposals. When process

[1]The message exchange was the time for a single process to send a message to nine other processes and receive all acknowledgements. The only function of the nine processes was to immediately reply to any message received (all of which were less than 5 bytes). Datagram sockets were used with all processes on seperate hosts (parallelizing the test) and connected via 100Mbps ethernet channels with a normal load.

failures occur during an agreement, the latency can be become extreme. With optimism, the idea is to remove this latency in either case.

To implement optimism, let $\mathcal{S}$ be the set of binary proposals and N be the number of processes participating in the agreement. The functions *opt*() and *com*() can be easily defined.

$$opt(\mathcal{S}) = \begin{cases} \text{NULL} & \text{if } \mathcal{S} = \text{NULL} \\ 0 & \text{if majority of proposals in } \mathcal{S} = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$com(\mathcal{S}) = \begin{cases} 0 & \text{if majority} = 0 \text{ and } |\mathcal{S}| = N \\ 1 & \text{if majority} \neq 0 \text{ and } |\mathcal{S}| = N \\ \text{NULL} & \text{otherwise} \end{cases}$$

It can be easily seen that these definitions satisfy our properties for *opt*() and *com*() since *opt*() always returns a value when $\mathcal{S}$is not NULL, *com*() only does so when a process can commit, and $opt() = com()$ when $com() \neq$ NULL.

4.2 Approximate agreement

Approximate agreement was presented by Dolev, et. al. in [6]. This algorithm guarantees agreement to within ϵ for every correct process, where ϵ can be defined to be as small as desired. The algorithm operates in a completely asynchronous environment in which t process crashes can occur. Approximate agreement requires that the number of processes participating in the agreement, N, be at least $5t+1$ and that messages are reliable. The algorithm operates in rounds with the number of rounds required, r_{total}, determined in the first round. A process waits until it has received $N-t$ proposals for round r, which are put in the set V_r. At round $r+1$ a process proposes a new value v_r calculated as follows:

$$v_r = f_{2t,t}(V_r) = \text{mean}(\text{select}_{2t}(\text{reduce}^t(V_r)))$$

That is, the set is truncated by using the *reduce* function to remove the first t and last t elements. The set is further reduced by the *select* function which returns a set with the first item and every $2t^{th}$ item in the set. The mean is then taken of this selected set to close in on the final agreement value.

The problem of sensor agreement lends itself to optimism, because when sensors are correct, they will generally already agree within ϵ. Therefore, by having a process optimistically execute with its own sensor value, it is very likely to perform useful work while the agreement algorithm is running.

To use optimism, we let $\mathcal{S}_r = V_r$, $\mathcal{S} = \cup_r \mathcal{S}_r$, and to agree within ϵ we set r_{total} in the first round by specifying agreement within $\frac{\epsilon}{3}$ (we assume $r_{total} \neq 0$). We use $\frac{\epsilon}{3}$ because whenever agreement occurs, this allows optimistic execution to have varied from the final committed value at a process by $\frac{\epsilon}{3}$. Forcing agreement to $\frac{\epsilon}{3}$ but allowing this same variation of optimism, ensures agreement within ϵ. *Note that this may require additional rounds.* The variable current_val (0 initially) is set to $opt(\mathcal{S})$ whenever the return value changes. Let $v_r = \text{mean}(\text{select}_{2t}(\text{reduce}^t(\mathcal{S}_r)))$, which is also the proposal for round $r+1$. The functions for optimsim are defined as:

$$opt(\mathcal{S}) = \begin{cases} \text{initial proposal} & \text{before round 0 completes} \\ \text{current\_val} & \text{at completion of round } r \text{ and } \mid v_r - \text{ current\_val } \mid \leq \frac{\epsilon}{3} \\ v_r & \text{otherwise} \end{cases}$$

$$com(\mathcal{S}) = \begin{cases} \text{current\_val} & \text{at completion of round } r_{total} \text{ and } \mid v_{r_{total}} - \text{ current\_val } \mid \leq \frac{\epsilon}{3} \\ v_{r_{total}} & \text{at completion of round } r_{total} \text{ and } \mid v_{r_{total}} - \text{ current\_val } \mid > \frac{\epsilon}{3} \\ \text{NULL} & \text{otherwise} \end{cases}$$

Note that $opt(\mathcal{S})$ will return NULL for $r = 0$ if no proposal has been made by the process. Using this definition, during round 0, a process will optimistically assume that its own proposal is correct. In all succeeding rounds, $opt(\mathcal{S})$ will only change if a new value for that round differs from current_val by greater than $\frac{\epsilon}{3}$. We also define $com(\mathcal{S})$ in this manner. The final value calculated from $\mathcal{S}_{r_{total}}$ will only be used if it differs from current_val by greater than $\frac{\epsilon}{3}$. Since the algorithm agrees to within $\frac{\epsilon}{3}$, the values returned by $com(\mathcal{S})$ for all processes will agree within ϵ.

We now verify that our choice of $opt()$ and $com()$ satisfy the properties given earlier.

Theorem 4.1 *$opt(\mathcal{S})$ for sensor agreement satisfies maximum and valid optimism.*

Proof: Maximum optimism:
Referring to the optimistic algorithm, $\mathcal{S}$ becomes non NULL whenever the current process makes a proposal, at which time $r = 0$, or the process receives a messages with $\mathcal{S}$ non NULL (in which case $r \geq 0$). Therefore, $\mathcal{S}$ is only non NULL if $r \geq 0$. For $r = 0$, $opt(\mathcal{S})$ returns the process's own proposal. If $r > 0$, $opt(\mathcal{S})$ returns current_val (in which case current_val $\neq$ NULL, otherwise $\mid v_r -$ current_val $\mid \nleq \frac{\epsilon}{3}$), or v_r.
Valid optimism:
For $com(\mathcal{S})$ to return a value requires that $r = r_{total} > 0$. If $\mid v_r -$ current_val $\mid \leq \frac{\epsilon}{3}$ then $com(\mathcal{S})$ $= opt(\mathcal{S}) =$ current_val. Otherwise $com(\mathcal{S}) = opt(\mathcal{S}) = v_{r_{total}}$. □

Theorem 4.2 *$com(\mathcal{S})$ for sensor agreement satisfies safety and validity.*

Proof: Safety:
We note that whenever the number of rounds reaches r_{total}, approximate agreement will return a value. $com(\mathcal{S})$ will not return a value until $r = r_{total}$.
Validity:
Approximate agreement will return $v_{r_{total}}$ when $r = r_{total}$. $com(\mathcal{S})$ will return $v_{r_{total}} \pm \frac{\epsilon}{3}$. Now we show that $\sigma = \frac{\epsilon}{3}$. Since the goal is to have all processes agree within ϵ of each other if all $v_{r_{total}}$ agree within $\frac{\epsilon}{3}$, then at any process the agreement value can vary from $v_{r_{total}}$ by $\frac{\epsilon}{3}$. Therefore, $\sigma = \frac{\epsilon}{3}$. □

In section 5 we quantify the amount of time optimism is useful under certain conditions.

4.3 Leader election

We define a simple leader election protocol among N processes for an asynchronous environment in which at most $\frac{N}{2}$ processes can fail. Messages are again assumed to be reliable. Every process can suspect every other process using some kind of failure detector. In addition to suspicion, a failure detector may remove a process from suspicion. The protocol, then, works as follows in a faulty environment:

1. Every process begins with the same predefined leader and leader succession table.
2. If a process suspects the leader it informs every other process of this suspicion and nominates the next unsuspected process in its succession table. Upon receiving a suspicion message, a process can return an acknowledgement, if it agrees, or a negative acknowledgement if not.
3. If the suspecting process receives at least $\frac{N}{2}$ positive acknowledgements it broadcasts this information to all other processes. Any process receiving this message uses this nominated process as the new leader.
4. If the suspecting process receives at least $\frac{N}{2}$ negative acknowledgements, it desists from initiating the leader election algorithm for some predetermined time.

For liveness we assume that any process will initiate the algorithm again after some local time if no outcome is observed. We use (p, q) to denote that process p suspects process q. For leader election, the processes use $\mathcal{S}$ as defined by the optimistic algorithm to collect suspicions. $opt(\mathcal{S})$ and $com(\mathcal{S})$, however, use the set $\mathcal{S}_L$ where L is the current leader:

$$\mathcal{S}_L = \{(p, L) \mid (p, L) \in \mathcal{S} \wedge L \text{ suspected by local process}\}$$

Therefore, if the local process does not suspect L, then $\mathcal{S}_L$ for that process is empty.

We define a successor function successor(p) as returning the successor of p using the succession table. We now give the $com()$ and $opt()$ functions.

$$com(\mathcal{S}_L) = \begin{cases} \text{successor}(L) & \text{if } \mid \mathcal{S}_L \mid > \frac{N}{2} \\ \text{NULL} & \text{otherwise} \end{cases}$$

$$opt(\mathcal{S}_L) = \begin{cases} \text{successor}(L) & \text{if } \mathcal{S}_L \neq \{\} \\ \text{NULL} & \text{otherwise} \end{cases}$$

When $com(\mathcal{S}_L)$ returns a value, the scheme can assign $L = com(\mathcal{S}_L)$. We show that our choice of $opt(\mathcal{S}_L)$ and $com(\mathcal{S}_L)$ satisfy the properties given earlier.

Theorem 4.3 *$opt(\mathcal{S}_L)$ for leader election satisfies maximum and valid optimism.*

Proof: Maximum optimism:
Obvious, by definition of $opt(\mathcal{S}_L)$.
Valid optimism:
If $com(\mathcal{S}_L) \neq$ NULL, then $\mathcal{S}_L \neq \{\}$. In this case, both $com(\mathcal{S}_L)$ and $opt(\mathcal{S}_L)$ will equal successor(L).
□

Theorem 4.4 *$com(\mathcal{S}_L)$ for leader election satisfies safety and validity.*

Proof: Note that the leader election algorithm at a P_i will commit to a new leader when P_i knows of greater than $\frac{N}{2}$ processes that suspect the current leader. Therefore, if a set $\mathcal{C}$exists such that it contains $> \frac{N}{2}$ entries of the form (p, L), then leader election will choose the successor of L as the new leader. Note that when a value is returned by $com(\mathcal{S}_L)$, $\mathcal{C} \subseteq \mathcal{S}$and the leader returned is the successor of L. □

It is noted that a process suspicion usually is correct under most operating conditions [10]. When a leader does fail, the optimistic protocol will allow useful work to occur during the two message delay required to elect the leader as shown in figure 1a.

The problem of optimism and leader election can also be considered in a system in which partitions can occur. Under this condition, it may be desireable to have processes optimistically choose a leader within a partition (even if the partition contains a minority of processes). Doing this can allow the processes in each partition to coordinate with their local leader. The major problem with this is what to do upon remerging. This is dependent on the application. For example, it may be desireable to commit some of the work and rollback other work.

We add a rule for optimistic execution which allows processes to optimistically choose a leader in a small partition. First, we define $\text{successor}_k(L)$ to be the kth successor of L.

> **Rule for partitioning:** If $\forall k : 0 \geq k \geq j : \mathcal{S}_{successor_k(L)} \neq \{\}$, then optimism uses $opt(\mathcal{S}_{\text{successor}_j(L)})$.

Note that $\text{successor}_0(L) = L$. This use of $opt(\mathcal{S}_L)$ will cause a process to optimistically elect a new leader whenever the current optimistic leader is suspected. This is useful in a partition in which a process may get cut off from several other processes simultaneously. When remerging of the partitions occur and $\mathcal{S}_{\text{successor}_k(L)}$ becomes empty, optimism will revert back to the same value as $com(\mathcal{S}_L)$. However, since it may be desireable to maintain work performed under an optimistic leader, another rule should be added to detect remerging and allow for keeping optimistic execution.

> **Rule for remerging:** If optimism has been using $opt(\mathcal{S}_{\text{successor}_j(L)})$ and later $\mathcal{S}_{\text{successor}_k(L)}$ becomes empty for $k < j$, do not rollback if optimistic_execution_useful().

This rule would allow optimistic execution to be saved if it was determined that it did not cause adverse affects. The definition of the function optimistic_execution_useful() would define how the optimistic execution was interpreted.

In a partitioned network, the optimistic protocol will produce two types of behavior depending upon whether a process is in the same partition as the next alive leader or not. If a process is in the partition of the next successor which is alive, then the process will correctly choose this process as its next leader. If this partition has less than $\frac{N}{2}$ processes, optimism will be required to continue with useful work. In any other partitions, the optimistic algorithm will cause the members of each partition to optimistically choose a member among themselves as the next leader. Since the elected leader in these partitions will also be contained in them, potentially useful work can be performed. However, since the execution is optimistic, this work cannot be committed. Once two or more partitions re-merge $\mathcal{S}_L$ will shrink at each process as the processes learn that some of the suspected processes are not dead. Notice that this will not necessarily cause $opt(\mathcal{S}_L)$ to choose a new leader. Instead, $com(\mathcal{S}_L)$ may eventually return a new leader. By the added rules for partitioning and remerging, individual processes can decide whether optimistic execution can be saved. Proofs of this operation are not given here.

5 Quantifying useful optimism

In this section we quantify the amount of useful optimism that can be achieved in our implementations of binary agreement, approximate agreement, and leader election.

5.1 Binary agreement

For the binary agreement problem, let there be N processes where k of these choose the same value and are a majority. For simplicity we will assume that the majority value is 1.

First we will consider optimism where proposal messages are received at a process at about the same time. That is, there is a delay from agreement start until these proposals are received. It is obvious that if this receive pattern occurs across all processes that $\frac{k}{N}$ ($> 50\%$) processes, will perform worthwhile optimism. This is in contrast to an optimized majority agreement algorithm which simply waits to receive a majority of either value. Such an optimized algorithm would have to wait until the point at which messages are received, that is, no benefit would be realized. Although this assumption of receives produces fairly good results, experiments show that the round trip message time is not the major component of latency, but rather the time to receive. The next scenario we give reflects this.

Next we consider when proposals are received, on average, evenly spaced in time at a process. Since there are k proposals with value 1, there are $\binom{N}{k}$ different orders in which these proposals can be received. After every receive (and the initial choice of its own proposal) every process will reevalutate its optimism. This is as given in our definition of *opt*() for binary agreement. A majority of any vote will cause it to choose that value while equal numbers of proposals will cause it to randomly choose an optimistic value. If a process has received i proposals (including its own) the probability that j of these are 1 is:

$$\frac{\binom{i}{j}\binom{N-i}{k-j}}{\binom{N}{k}}$$

Now, as just stated, if $j \geq \frac{i}{2}$ then the process will choose 1 optimistically with probability 1. We assume that when a process must roll back, it maintains the data structure of variables for that value (so it can restart optimism, in case this value is again chosen optimistically). Then, the fraction of agreement time spent executing correct optimism is:

$$\frac{1}{N-1}\sum_{i=1}^{N-1}\sum_{j=max(\lceil\frac{i}{2}\rceil,k-N+i)}^{min(i,k)}\left[\frac{\binom{i}{j}\binom{N-i}{k-j}}{\binom{N}{k}}\right.$$

For $N = 5$ and $k = 3$ and 4, the fraction of time spent executing correct optimism is $\frac{13}{20}$ and $\frac{9}{10}$. However, for the optimized majority algorithm described above, on average it could only commit $\frac{1}{8}$th and $\frac{7}{20}$th earlier than the base algorithm respectively. We note that as $k \rightarrow \frac{N}{2}$, the fraction of useful optimism approaches $\frac{1}{2}$. Therefore, optimism is always useful for at least 50% of the agreement time.

5.2 Approximate agreement

For approximate agreement we first consider that the optimistic implementation must calculate rounds based on $\frac{\epsilon}{3}$ rather than ϵ. This means the optimistic approach will require extra rounds.

Let r_{total} and $r_{opttotal}$ be the number of rounds required for approximate agreement and approximate agreement with optimism. From [6], the number of rounds required by the algorithm (calculated individually at each process) is:

$$r_{total} = \lceil log_{(\lfloor \frac{N-3t-1}{2t} \rfloor + 1)} \frac{\delta(V_0)}{\epsilon} \rceil$$

where $\delta(V_0)$ is $\max(V_0) - \min(V_0)$ and V_0 is the first $N-t$ proposals received in the initial round. From this equation, the number of extra rounds required on average for our optimistic implementation is:

$$r_{optextra} = log_{(\lfloor \frac{N-3t-1}{2t} \rfloor + 1)} 3.$$

For $n = 5t+1$, this is 1.58. It is obvious that to have our optimistic implementation worthwhile, every process should be operating with correct optimism at least by the time it would have committed without optimism. Let $r_{optuseful}$ be the average number of rounds that processes execute with correct optimism (that is, with a committable value) *before* r_{total}. Therefore, to be worthwhile, $r_{optuseful}$ should be greater than 0. Note that the total number of rounds that each process actually executes optimistically is $r_{optuseful} + r_{optextra}$.

From [6], U is the set of proposals from all N processes. V and W are the sets received by processes p and q respectively and used to determine the next round. Then, $\mid U \mid = N$ and $\mid V \mid = \mid W \mid = N - t$. Let U_0 be the set of initial proposals. In the worst case, $\delta(V_0) = \delta(U_0)$, when there are greater than t of the highest and lowest values of U_0.

Again, from [6], at each round r:

$$\mid f_{2t,t}(W_r) - f_{2t,t}(V_r) \mid \leq \frac{\delta(U_r)}{c(N-3t, 2t)}$$

where $c(i,j) = \lfloor \frac{i-1}{j} \rfloor + 1$. Suppose that the average number of rounds calculated by each process is r_{avg}. Then,

$$r_{avg} = log_{(\lfloor \frac{N-3t-1}{2t} \rfloor + 1)} \frac{\delta(A_0)}{\epsilon}$$

where $\delta(A_0)$ is the average difference of minimum and maximum values in the initial round among all processes.

Now, suppose that agreement among sensors is being performed in which the sensors will most of the time report any value between x and $x + \Delta$ with equal probability (x is unknown). In this case, non-faulty sensors will be evenly distributed, with $\delta(U_0) = \Delta$, but without a guarantee of this distribution. Under our assumptions, the majority of the time, $\mid f_{2t,t}(W_0) - f_{2t,t}(V_0) \mid \leq \delta(U_0)/(N-1)$ since there are a maximum of N proposals. Note also, that $\delta(A_0) = \delta(U_0)\frac{(N-t-1)}{(N-1)}$. Because of the likely distribution of the sensors, assume that ϵ has been chosen such that, $\frac{\epsilon}{3} = \frac{\Delta}{N-1} = \frac{\delta(U_0)}{N-1}$. This means that most of the time, all processes will be able to execute with correct optimism for all rounds after the initial round. From this, we see that $\epsilon = \frac{3\delta(U_0)}{N-1}$. This leads to the following equation for r_{avg}:

$$r_{avg} = log_{(\lfloor \frac{N-3t-1}{2t} \rfloor + 1)} \frac{N-t-1}{3}$$

If we use $N = 5t+1$, and choose $t = 3$, $r_{avg} = 2$ rounds, and therefore $r_{optuseful} = 2$ rounds. That is, 2 rounds of extra useful computation on average over normal approximate agreement (the base algorithm). $r_{optuseful}$ increases with greater t and decreases with less t.

For sensor and other types of agreement in which an initial distribution is known to likely occur, similar analyses can be made. In this way, it can be determined if choosing ϵ based on the distribution will make it beneficial to use optimism. The other consideration must be the extra rounds required to commit.

5.3 Leader election

Combining optimism with leader election requires very little analysis. We consider the case in which partitioning does not occur. Since a new leader is only elected in our algorithm when the current leader is suspected, we can use the probability of correct suspicion. Let p_{dead} be the probability that a suspected process cannot refute this suspicion. In real world applications, p_{dead} is close to 1. That is, suspicion of a crash usually is correct. Since optimism assumes that the current leader is crashed when suspected or told to suspect, optimism across all processes is accurate (or useful) with probability p_{dead}. Therefore, the amount of useful execution is p_{dead} times the time required to elect a new leader.

6 Experimental results

We have implemented optimism for the problems of binary and approximate agreement. In this section we discuss our experimental results.

6.1 Binary Agreement

For binary agreement we implemented the algorithm such that the environment appears error-free: processes recover and messages are resent if lost. We used a set of twenty processes for the agreement which all proposed either a 1 or 0. Whenever the number of 1s and 0s was the same a process chose 1. We varied the probability of choosing 1 from 50 to 80 percent, performing twenty runs each for the probabilities 50, 60, 70, and 80. We did this for both the cumulative approach and non-cumulative approach. Figure 3 show the results. As expected the cumulative approach did better, however both are similar when the probability of a 1 is 70 or 80 percent. With a probability of 50, the percent of non-cumulative useful optimism is actually closer to 60 rather than 50. This is because of our breaking ties by choosing 1, and in general keeping proposals of value 1 as the majority.

6.2 Approximate agreement

For our implementation of approximate agreement, figure 4 shows the results for both uniform (as discussed in the last section) and normal distributions of the proposal inputs. We used $N = 6$ and $t = 1$. The graphs plot the average round at which correct optimism ($opt(\mathcal{S}){=}com(\mathcal{S})$) began, as well as r_{total} and $r_{opttotal}$. $r_{optuseful}$ is measured from the start of correct optimism to r_{total}. The average rounds are versus the fraction $\frac{\delta(U_0)}{\epsilon}$, where U_0 is the set of all initial proposals. We performed 20 runs for each of $\frac{\delta(U_0)}{\epsilon} = 10, 30, 100, 300, 1000$ with the average for each plotted. The graphs show that $r_{optuseful}$ can be as large as 11 rounds. Also note that, as predicted, on average $r_{opttotal} - r_{total} = 1.5$.

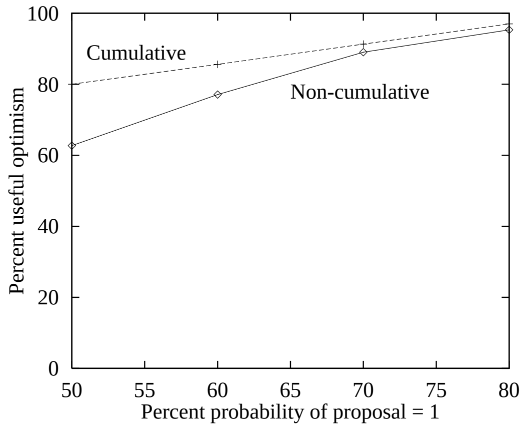


Figure 3: Non-cumulative and cumulative optimism in binary agreement

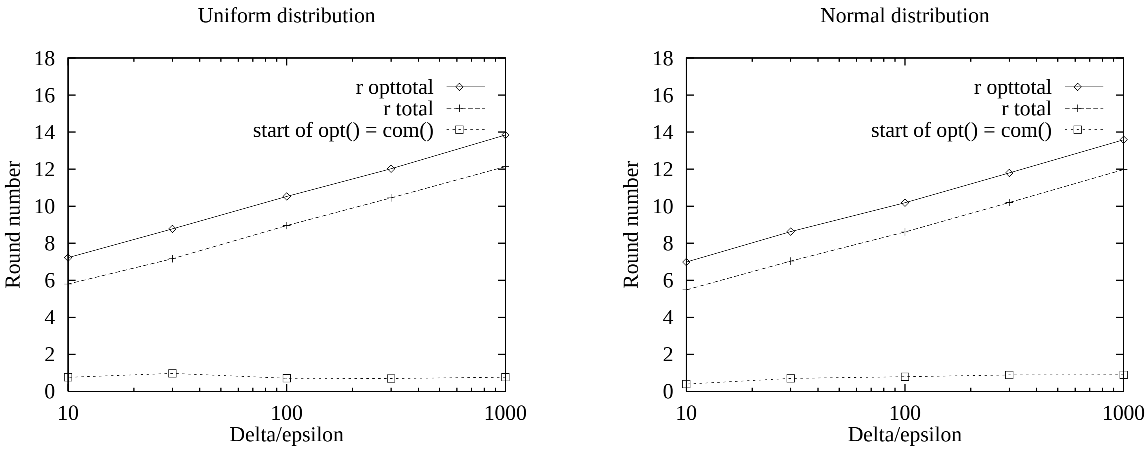


Figure 4: Optimistic, approximate agreement with uniform and normal distributions of proposals

7 Discussion of tunable optimism

In our definition of of the function *opt*(), we defined the property *maximum optimism.* This property requires the function to return a value whenever $\mathcal{S}$ is not empty. Our applications and analyses of the optimistic algorithm was then built on this property. There may be cases in which it is desirable to delay optimism. For example, suppose that the outcome of a particular agreement problem can be much more accurately predicted after k proposals have been received, where $1 \geq kgeqN$. It may be desireable to forgo optimistic execution in such a case, until after k proposals have been received to perform other tasks. We call k a tunable parameter and note that when $k = 1$ the optimistic delivers maximum optimism as given earlier. When k is such that $opt() = com()$, optimism is eliminated and the operation of the agreement reduces to the base algorithm.

8 Conclusions

We have presented an approach for reducing the latency associated with agreement in asynchronous distributed systems in both failure-free and faulty environments. The optimistic algorithm presented sits on top of any base agreement algorithm that would normally be used to solve the agreement problem. Then when optimism executes, proposal information is spread with all messages. Applications can begin executing using optimistic values before commit occurs. Our approach can be applied to general agreement problems, whereas previous work has focused on database transactions. We have given three examples of the use of optimism: binary agreement, approximate agreement, and leader election. We have also analyzed the usefulness of optimism and shown that our algorithm can deliver additional useful execution the majority of the time.

References

[1] D. Agrawal, A. Bernstein, "A Nonblocking Quorum Consensus Protocol for Replicated Data," *IEEE Transactions on Parallel and Distributed Systems,* Vol. 2, No. 2, April 1991, pp. 171-179.

[2] M. Ben-Or, "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract)," *Proceedings of the 2nd ACM Annual Symposium on Principles of Distributed Computing*, Montreal, Quebec, 1983, pp. 27-30.

[3] T. Chandra, S. Toueg, "Unreliable Failure Detectors for Asynchronous Systems (Preliminary Version)," *Proceedings of the 10th ACM Annual Symposium on Principles of Distributed Computing*, August, 1991, pp. 325-340.

[4] T. Chandra, V. Hadzilacos, S. Toueg, "The Weakest Failure Detector for Solving Consensus," *Proceedings of the 11th ACM Annual Symposium on Principles of Distributed Computing,* Vancouver, B.C., 1992, pp. 147-158.

[5] S. Davidson, H. Garcia-Molina, D. Skeen, "Consistency in Partitioned Networks," *ACM Computing Surveys*, vol.17, no. 3, ACM Press, 1985, pp. 341-370.

[6] D. Dolev, N. Lynch, S. Pinter, E. Stark, W. Wiehl, "Reaching Approximate Agreement in the Presence of Faults," *Journal of the ACM*, Vol.33, no.3, July 1986, pp. 499-516.

[7] D. Dolev, D. Malki, R. String, "A Framework for Partitionable Membership Service," TR CS95-4, Institute of Computer Science, Hebrew University of Jerusalem, 1995.

[8] C. Dwork, N. Lynch, L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM*, vol. 35, no. 2, 1988, pp. 288-323.

[9] M. Fischer, N. Lynch, M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, vol. 32, no.2, 1985, pp.374-382.

[10] R. Friedman, I. Keidar, D. Malki, K. Birman, D. Dolev, "Deciding in Partitionable Networks," Technical report 95-1554, Dept. of Computer Science, Cornell University, 1995.

[11] A. Ricciardi, K. Birman, "Consistent Process Membership in Asynchronous Environments," in *Reliable Distributed Computing with the ISIS Toolkit*, IEEE Computer Society Press, 1994, pp. 237-262.