$= (1/n)|n\Delta_{pq} + V_{qp} + \Sigma_{i=3..n} V_{p_i p} - V_{pq} - \Sigma_{i=3..n} V_{p_i q}|$

$= (1/n)|(\Delta_{pq} + V_{qp}) + (\Delta_{pq} - V_{pq}) + \Sigma_{i=3..n}(\Delta_{pq} + V_{p_i p} - V_{p_i q})|$

$\leq (1/n)(|\Delta_{pq} + V_{qp}| + |\Delta_{pq} - V_{pq}| + \Sigma_{i=3..n}|\Delta_{pq} + V_{p_i p} - V_{p_i q}|)$

$\leq (1/n)(\varepsilon + \varepsilon + \Sigma_{i=3..n}|\Delta_{pq} + V_{p_i p} - V_{p_i q}|)$, by Lemmas 3-6 and 3-4

$= (1/n)(2\varepsilon + \Sigma_{i=3..n}|\Delta_{pp_i} + \Delta_{p_i q} + V_{p_i p} - V_{p_i q}|)$, by Lemma 3-5

$= (1/n)(2\varepsilon + \Sigma_{i=3..n}|(V_{p_i p} - \Delta_{p_i p}) - (V_{p_i q} - \Delta_{p_i q})|)$, by Lemma 3-4

$\leq (1/n)(2\varepsilon + \Sigma_{i=3..n}|V_{p_i p} - \Delta_{p_i p}| + \Sigma_{i=3..n}|-(V_{p_i q} - \Delta_{p_i q})|)$

$\leq (1/n)(2\varepsilon + \Sigma_{i=3..n}\varepsilon + \Sigma_{i=3..n}\varepsilon)$, by Lemma 3-6

$\leq (1/n)(2\varepsilon + (n-2)2\varepsilon)$

$= 2\varepsilon(1 - 1/n)$. ∎

3.4.4 Validity

The validity result states that each new logical clock is within ε of what one of the initial logical clocks would have been.

Theorem 3-8: (Validity) Algorithm 3-1 bounds the adjustment within ε.

Proof: By definition, the amount to be added to $CORR_p$ is $A_p = (1/n)\,\Sigma_{q \in P} V_{qp}$. Then $\min_{q \in P} V_{qp} \leq A_p \leq \max_{r \in P} V_{rp}$. Let q be the process with the minimum V_{qp}. Let r be the process with the maximum V_{rp}. Then,

$V_{qp} \leq A_p \leq V_{rp}$.

By applying Lemma 3-6 to each end of this inequality, we get

$\Delta_{qp} - \varepsilon \leq V_{qp} \leq A_p \leq V_{rp} \leq \Delta_{rp} + \varepsilon$.

Adding p's initial clock value $C^0_p(t)$ for $t \geq t_f$, we get

$C^0_p(t) + \Delta_{qp} - \varepsilon \leq C^0_p(t) + A_p \leq C^0_p(t) + \Delta_{rp} + \varepsilon$,

which together with the definition of Δ implies

$C^0_q(t) - \varepsilon \leq L_p(t) \leq C^0_r(t) + \varepsilon$. ∎

Chapter Four

Maintenance Algorithm

4.1 Introduction

This chapter consists of an algorithm to keep synchronized clocks that are close together initially, and an analysis of its performance concerning how closely the clocks are synchronized and how close the clocks stay to real time. The algorithm handles clock drift and arbitrary process faults. The algorithm requires the clocks to be initially close together and less than one third of the processes to be faulty. (Dolev, Halpern and Strong [2] show that it is impossible without authentication to synchronize clocks unless more than two thirds of the processes are nonfaulty.)

This algorithm runs in rounds, resynchronizing periodically to correct for clock drift, and using a fault-tolerant averaging function based on those in [1] to calculate an adjustment. The size of the adjustment is independent of the number of faulty processes. At each round, n^2 messages are required, where n is the total number of processes. The closeness of synchronization achieved depends only on the initial closeness of synchronization, the message delivery time and its uncertainty, and the drift rate. We give explicit bounds on how the difference between the clock values and real time grows as time proceeds. The algorithm can be easily adapted to include reintegration of repaired processes as described in Section 4.8.

4.2 Problem Statement

We are now considering the situation in which clocks can drift slightly and some proportion of the processes can be faulty. Therefore, the statement of the problem differs from that in Chapter 3.

For a very small constant $\rho > 0$, we define a clock C to be *ρ-bounded* provided that for all t

$$1 - \rho \leq 1/(1 + \rho) \leq dC(t)/dt \leq 1 + \rho \leq 1/(1 - \rho).$$

We make the following assumptions:

1. All clocks are ρ-bounded, including those of faulty processes, i.e., the amount by which a clock's rate is faster of slower than real time is at most ρ. (Since faulty processes are permitted to take arbitrary steps, faulty clocks would not increase their

power to affect the behavior of nonfaulty processes.)

2. There are at most f faulty processes, for a fixed constant f, and the total number of processes in the system, n, is at least 3f + 1.

3. A START message arrives at each process p at time T^0 on its initial logical clock C^0_p, and t^0_p is the real time when this occurs. Furthermore, the initial logical clocks are closely synchronized, i.e., $|c^0_p(T^0) - c^0_q(T^0)| \leq \beta$, for some fixed β and all nonfaulty p and q.

We let $tmax^0 = max_{p \text{ nonfaulty}}\{t^0_p\}$ and analogously for $tmin^0$.

The object is to design an algorithm for which every execution in which the assumptions above hold satisfies the following two properties.

1. γ-Agreement: $|L_p(t) - L_q(t)| \leq \gamma$, for all $t \geq tmin^0$ and all nonfaulty p, q.

2. $(\alpha_1,\alpha_2,\alpha_3)$-Validity: $\alpha_1(t - tmax^0) + T^0 - \alpha_3 \leq L_p(t) \leq \alpha_2(t - tmin^0) + T^0 + \alpha_3$, for all $t \geq t^0_p$ and all nonfaulty p.

The Agreement property means that all the nonfaulty processes are synchronized to within γ. The Validity property means that the local time of a nonfaulty process increases in some relation to real time. We would, of course, like to minimize α_1, α_2, α_3, and γ.

4.3 Properties of Clocks

We give several straightforward lemmas about the behavior of (ρ- bounded) clocks.

Lemma 4-1: Let C be any clock.

(a) If $t_1 \leq t_2$, then

$$(1 - \rho)(t_2 - t_1) \leq (t_2 - t_1)/(1 + \rho) \leq C(t_2) - C(t_1) \leq (1 + \rho)(t_2 - t_1) \leq (t_2 - t_1)/(1 - \rho).$$

(b) If $T_1 \leq T_2$, then

$$(1-\rho)(T_2 - T_1) \leq (T_2 - T_1)/(1 + \rho) \leq c(T_2) - c(T_1) \leq (1 + \rho)(T_2 - T_1) \leq (T_2 - T_1)/(1-\rho).$$

Proof: Straightforward. ∎

Lemma 4-2: Let C and D be clocks.

(a) If $dC(t)/dt = 1$ and $T_1 \leq T_2$, then

$$|(c(T_2) - d(T_2)) - (c(T_1) - d(T_1))| = |(c(T_2) - c(T_1)) - (d(T_2) - d(T_1))| \leq \rho(T_2 - T_1).$$

(b) If $T_1 \leq T_2$, then

$|(c(T_2) - d(T_2)) - (c(T_1) - d(T_1))| = |(c(T_2) - c(T_1)) - (d(T_2) - d(T_1))| \leq 2\rho(T_2 - T_1)$.

(c) If $dC(t)/dt = 1$ and $t_1 \leq t_2$, then

$|(C(t_2) - D(t_2)) - (C(t_1) - D(t_1))| = |(C(t_2) - C(t_1)) - (D(t_2) - D(t_1))| \leq \rho(t_2 - t_1)$.

(d) If $t_1 \leq t_2$, then

$|(C(t_2) - D(t_2)) - (C(t_1) - D(t_1))| = |(C(t_2) - C(t_1)) - (D(t_2) - D(t_1))| \leq 2\rho(t_2 - t_1)$.

Proof: Straightforward using Lemma 4-1. ∎

Lemma 4-3: Let C and D be clocks, $T_1 \leq T_2$. Assume $|c(T) - d(T)| \leq \alpha$ for all T, $T_1 \leq T \leq T_2$. Let $t_1 = \min\{c(T_1), d(T_1)\}$ and $t_2 = \max\{c(T_2), d(T_2)\}$.

Then $|C(t) - D(t)| \leq (1 + \rho)\alpha$ for all t, $t_1 \leq t \leq t_2$.

Proof: There are four cases, which can easily be shown to be exhaustive.

Case 1: $c(T_1) \leq t \leq c(T_2)$.

Let $T_3 = C(t)$, so that $T_1 \leq T_3 \leq T_2$. By hypothesis, $|c(T_3) - d(T_3)| \leq \alpha$. Then $|T_3 - D(t)| \leq (1 + \rho)\alpha$, by Lemma 4-1.

Case 2: $d(T_1) \leq t \leq d(T_2)$. This case is analogous to the first.

Case 3: $c(T_2) < t < d(T_1)$.

Then $c(T_1) < t < d(T_1)$. So $C(t) > D(t)$, and thus

$|C(t) - D(t)| = C(t) - D(t) = (C(t) - T_1) + (T_1 - D(t))$

$\leq (1 + \rho)(t - c(T_1)) + (1 + \rho)(d(T_1) - t)$, by Lemma 4-1,

$= (1 + \rho)(d(T_1) - c(T_1)) \leq (1 + \rho)\alpha$.

Case 4: $d(T_2) < t < c(T_1)$. This case is analogous to the third. ∎

4.4 The Algorithm

4.4.1 General Description

The algorithm executes in a series of rounds, the i-th round for a process triggered by its logical clock reaching some value T^i. (It will be shown that the logical clocks reach this value within real time β of each other.) When any process p's logical clock reaches T^i, p broadcasts a T^i message. Meanwhile, p collects T^i messages from as many processes as it can, within a particular bounded amount of time, measured on its logical clock. The bounded amount of time is of length $(1 + \rho)(\beta$

$+ \delta + \varepsilon$), and is chosen to be just large enough to ensure that T^i messages are received from all nonfaulty processes. After waiting this amount of time, p averages the arrival times of all the T^i messages received, using a particular fault-tolerant averaging function. The resulting average is used to calculate an adjustment to p's correction variable, thereby switching p to a new logical clock.

The process p then waits until its new clock reaches time $T^{i+1} = T^i + P$, and repeats the procedure. P, then, is the length of a round in local time.

The fault-tolerant averaging function is derived from those used in [1] for reaching approximate agreement. The function is designed to be immune to some fixed maximum number, f, of faults. It first throws out the f highest and f lowest values, and then applies some ordinary averaging function to the remaining values. In this paper, we choose the midpoint of the range of the remaining values, to be specific.

4.4.2 Code for an Arbitrary Process

Global constants: ρ, β, δ, ε, and P, as defined above.

Local variables:

- CORR, initially arbitrary; correction variable which corrects physical time to logical time.
- ARR[q], initially arbitrary; array containing the arrival times of the most recent messages, one entry for each process q.
- T, initially undefined; local time at which the process next intends to send a message.

Conventions:

- NOW stands for the current logical clock time (i.e., the physical clock reading + CORR). NOW is assumed to be set at the beginning of a step, and cannot be assigned to.
- REDUCE, applied to an array, returns the multiset consisting of the elements of the array, with the f highest and f lowest elements removed.
- MID, applied to a multiset of reals numbers, returns the midpoint of the set of values in the multiset.

The code is in Figure 4-1.

```
beginstep(u)
do forever

/* in case T^i messages are received before this process reaches T^i */

     while u = (m,q) for some message m and process q do
          ARR[q] := NOW
          endstep
          beginstep(u)
          endwhile

/* fall out of the loop when u = START or TIMER; begin round */

     T := NOW
     broadcast(T)
     set-timer(T + (1 + ρ)(β + δ + ε))

     while u = (m,q) for some message m and process q do
          ARR[q] := NOW
          endstep
          beginstep(u)
          endwhile

/* fall out of the loop when u = TIMER; end round */

     AV := mid(reduce(ARR))
     ADJ := T + δ - AV
     CORR := CORR + ADJ
     set-timer(T + P)
     endstep
     beginstep(u)
     enddo
```

Figure 4-1: Algorithm 4-1, Maintaining Synchronization

4.5 Inductive Analysis

Although the algorithm is fairly simple, its analysis is surprisingly complicated and requires a long series of lemmas.

4.5.1 Bounds on the Parameters

We assume that the parameters ρ, δ, and ϵ are fixed, but that we have some freedom in our choice of P and β, subject to the reasonableness of our assumption that the clocks are initially synchronized to within β. We would like β to be as small as possible, to keep the clocks as closely synchronized as we can. However, the smaller β is, the smaller P must be (i.e., the more frequently we must synchronize).

There is also a lower bound on P. In order for the algorithm to work correctly, we need to have P sufficiently large to ensure the following.

(1) After a nonfaulty process p resets its clock, the local time at which p schedules its next broadcast is greater than the local time on the new clock, at the moment of reset.

(2) A message sent by a nonfaulty process q for a round arrives at a nonfaulty process p after p has already set its clock for that round.

Sufficient bounds on P turn out to be:

$P > 2(1 + \rho)(\beta + \varepsilon) + (1 + \rho)\max\{\delta, \beta + \varepsilon\} + \rho\delta$, and

$P \leq \beta/4\rho - \varepsilon/\rho - \rho(\beta + \delta + \varepsilon) - 2\beta - \delta - 2\varepsilon$.

A required lower bound on β is $\beta \geq 4\varepsilon + 4\rho(3\beta + \delta + 3\varepsilon) + 8\rho^2(\beta + \delta + \varepsilon)$.

Any combination of P and β which satisfies these inequalities will work in our algorithm. If P is regarded as fixed, then β, the closeness of synchronization along the real time axis, is roughly $4\varepsilon + 4\rho P$. This value is obtained by solving the upper bound on P for β and neglecting terms of order ρ.

4.5.2 Notation

Let $T^i = T^0 + iP$ and $U^i = T^i + (1 + \rho)(\beta + \delta + \varepsilon)$, for all $i \geq 0$.

For each i, every process p broadcasts T^i at its logical clock time T^i (real time t^i_p) and sets a timer to go off when its logical clock reaches U^i. When the logical clock reaches U^i (at real time u^i_p), the process resets its CORR variable, thereby switching to a new logical clock, denoted C^{i+1}_p. Also at real time u^i_p, the process sets a timer for the time on its physical clock when the new logical clock C^{i+1}_p reaches T^{i+1}. It is at least theoretically possible that this new timer might be set for a time on the physical clock which has already passed. If the timer is never set in the past, the process moves through an infinite sequence of clocks C^0_p, C^1_p, etc, where C^0_p is in force in the interval of real time $(-\infty, u^0_p)$, and each C^i_p, $i \geq 1$, is in force in the interval of real time $[u^{i-1}_p, u^i_p)$. If, however, the timer is set in the past at some u^i_p, then no further timers arrive after that real time, and no further resynchronizations occur. That is, C^{i+1}_p stays in force forever, and u^j_p and t^j_p are undefined for $j \geq i + 1$.

Let $tmin^i$ denote $\min_{p \text{ nonfaulty}}\{t^i_p\}$, and analogously for $tmax^i$, $umin^i$ and $umax^i$.

For p and q nonfaulty, let $ARR^i_p(q)$ denote the time of arrival of a T^i message from q to p, sent at q's clock time T^i, where the arrival time is measured on p's local clock C^i_p. (We will prove that C^i_p has actually been set by the time this message arrives.) Let AV^i_p denote the value of AV calculated by p using the ARR^i_p values, and let ADJ^i_p denote the corresponding value of ADJ calculated by p. Thus, $C^{i+1}_p = C^i_p + ADJ^i_p$.

This section is devoted to proving the following three statements for all $i \geq 0$:

(1) The real time t^i_p is defined for all nonfaulty p. (That is, timers are set in the future.)

(2) $|t^i_p - t^i_q| \leq \beta$, for all nonfaulty p and q. (That is, the separation of clocks is bounded by β.)

(3) $t^i_p + \delta - \varepsilon > u^{i-1}_q$, for all nonfaulty p and q, and $i \geq 1$. (That is, messages arrive after the appropriate clocks have been set.)

The proof is by induction. For $i = 0$, (1) and (2) are true by assumption and (3) is vacuously true.

Throughout the rest of this section, we assume (1), (2), and (3) hold for i. We show (1), (2), and (3) for $i + 1$ after bounding the size of the adjustment at each round.

4.5.3 Bounding the Adjustment

In this subsection, we prove several lemmas leading up to a bound on the amount of adjustment made by a nonfaulty process to its clock, at each time of resynchronization.

Lemma 4-4: Let p and q be nonfaulty.

(a) $ARR^i_p(q) \leq T^i + (1 + \rho)(\beta + \delta + \varepsilon)$.

(b) If $\delta - \varepsilon \geq \beta$, then $ARR^i_p(q) \geq T^i + (1 - \rho)(\delta - \varepsilon - \beta)$.

(c) If $\delta - \varepsilon \leq \beta$, then $ARR^i_p(q) \geq T^i - (1 + \rho)(\beta - \delta + \varepsilon)$.

Proof: Straightforward using Lemma 4-1. ∎

Lemma 4-5: Let p be nonfaulty. Then there exist nonfaulty q and r with

$ARR^i_p(q) \leq AV^i_p \leq ARR^i_p(r)$.

Proof: By throwing out the f highest and f lowest values, the process ensures that the remaining values are in the range of the nonfaulty processes' values. ∎

We are now able to bound the adjustment.