Chapter 12

TOLERATING FAULTS IN COUNTING NETWORKS

From the Proceedings of the 1999 Fault-Tolerant Parallel and Distributed Systems Workshop, in "Dependable Network Computing", by Kluwer Academic Publishing

Marc D. Riedel and Jehoshua Bruck
California Institute of Technology, 136-93, Pasadena, CA 91125

Abstract Counting networks were proposed by Aspnes, Herlihy and Shavit [3] as a low-contention concurrent data structure for multiprocessor coordination. We address the issue of tolerating faults in counting networks. In our fault model, balancer objects experience *responsive crash failures*: they behave correctly until they fail, and thereafter they are inaccessible. We propose two methods for tolerating such faults. The first is based on a construction of a *k-fault-tolerant balancer* with $2(k+1)$ bits of memory. All balancers in a counting network are replaced by fault-tolerant ones. Thus, a counting network with depth $O(\log^2 n)$, where n is the width, is transformed into a k-fault-tolerant counting network with depth $O(k \log^2 n)$.

We also consider the case where inaccessible balancers can be remapped to spare balancers. We present a bound on the error in the output token distribution of counting networks with remapped faulty balancers (a generalization of the error bound for sorting networks with faulty comparators presented by Yao & Yao [10]).

Our second method for tolerating faults is based on the construction of a *correction network*. Given a token distribution with a bounded error, the correction network produces a token distribution that is smooth (i.e., the number of tokens on each output wire differs by at most one – a weaker condition than the step property of counting networks). The correction network is constructed with fault-tolerant balancers. It is appended to a counting network in which faulty balancers are remapped to spare balancers. In order to tolerate k faults, the correction network has depth $2k(k+1)(\log n + 1)$, for a network of width n. Therefore, this method results in a network with a smaller depth provided that $O(k) < O(\log n)$. However, it is only applicable if it is possible to remap faulty balancers.

1 INTRODUCTION

Shared counting is the basis for many fundamental multiprocessor coordination algorithms, such as scheduling, load balancing and resource allocation. Such algorithms typically require that processes cooperate to assign consecutive integer values from a given range. The usual approach is to serialize access to a single shared counter value. However, due to high contention, accessing the counter value becomes a sequential bottleneck.

Counting networks were proposed by Aspnes, Herlihy and Shavit [3]. A counting network implements a mod n shared counter: in response to increment requests, processes are assigned counter values in the range $0, \ldots, n-1$. The counting network data structure consists of $O(n \log^2 n)$ balancer objects, each with a single bit of memory. Counting networks achieve high throughput by permitting multiple requests for counter values to proceed concurrently. Each request accesses only a small fraction of the balancers, so the contention on each balancer is low. Aspnes et al. give convincing experimental evidence that counting networks have higher throughput than conventional implementations when the load on the network is sufficiently high. For background information on counting networks and an explanation of the terminology used, the reader is referred to [3].

In this paper, we address the issue of tolerating faults in counting networks. Our fault model consists of dynamic failures in the counting network data structure. Specifically, we consider the case where balancers experience *responsive crash failures* [4]: they behave correctly until they fail, and thereafter they are inaccessible.

We propose two methods for tolerating such faults. The first is based on the construction of a *k-fault-tolerant balancer* with $2(k+1)$ bits of memory. All balancers in a counting network are replaced by fault-tolerant ones. Thus, a counting network with depth $O(\log^2 n)$, where n is the width, is transformed into a k-fault-tolerant counting network with depth $O(k \log^2 n)$.[1]

We also consider the case where inaccessible balancers can be *remapped* to spare balancers. A spare balancer is given a random initial state. If this state is different from the original balancer's state, the situation is equivalent to a spurious state transition. With remapped faulty balancers, the distribution of tokens at the output of a counting network may no longer satisfy the step property required for counting. We present an upper bound on the error in the output token distribution of

[1]All logarithms are base 2.

counting networks with faulty balancers. This is a generalization of the error bound for sorting networks with faulty comparators presented by Yao & Yao [10].

Our second method for tolerating faults is based on a construction of a *correction network*. Given a token distribution with a bounded error, the correction network produces a smooth output token distribution; that is, the number of tokens on each output wire differs by at most one. This is a weaker condition than the step property of counting networks; however, for applications such as load balancing it is sufficient. The correction network is constructed with fault-tolerant balancers. It is appended to a counting network in which faulty balancers are remapped to spare balancers. In order to tolerate k faults, the correction network has depth $2k(k+1)(\log n+1)$, for a network of width n. Therefore, this method results in a network with a smaller depth provided that $O(k) < O(\log n)$. However, it is only applicable if it is possible to remap faulty balancers.

2 FAULT MODEL

Several researchers have investigated failure models for shared-memory systems, and have proposed fault-tolerant constructions for shared objects [1][2][4][5]. In our fault model, balancers experience responsive crash failures:

Fault model: The memory location holding a balancer's state variable behaves correctly until it suffers an atomic failure. Thereafter, it is inaccessible.

Note that we do not consider process failures, which could result in lost tokens. Also, we do not consider errors affecting the network wiring information (the topology of the network is static).

Our strategy in coping with faults is to bypass inaccessible balancers. Thus, tokens are forwarded out along the same wire that they are received on (out the top if they are received on the top, or out the bottom if they are received on the bottom). Denote by x_t and x_b the number of tokens received prior to a fault on a balancer's top and bottom input wires, respectively; denote by x'_t and x'_b the number of tokens received after the fault on the balancer's top and bottom input wires, respectively. Denote by y_t and y_b the total number of tokens forwarded to its top and bottom output wires, respectively. We have

$$y_t = \left\lceil \frac{x_t + x_b}{2} \right\rceil + x'_t, \quad y_b = \left\lfloor \frac{x_t + x_b}{2} \right\rfloor + x'_b.$$

For what follows, we define the *distance* between two sequences $\mathbf{y} = y_0, y_1, \ldots, y_{n-1}$ and $\mathbf{y}' = y'_0, y'_1, \ldots, y'_{n-1}$ as one half the sum of the absolute value of the difference of their entries:

$$D(\mathbf{y}, \mathbf{y}') = \frac{1}{2}\sum_{i=0}^{n-1} |y_i - y'_i|.$$

3 TOLERATING FAULTS (1ST METHOD)

We describe a construction of a fault-tolerant balancer with $2(k+1)$ bits, capable of tolerating k faults. All balancers in a counting network are replaced by fault-tolerant ones. Thus, we transform a counting network with depth $O(\log^2 n)$ into a k-fault-tolerant counting network with depth $O(k \log^2 n)$. We note that similar results could be obtained based on the constructions for fault-tolerant shared objects presented by Afek et al. [2] and Jayanti et al. [4].

3.1 FAULT-TOLERANT BALANCER

In our construction, shown in Figure 1, a k-fault-tolerant balancer consists of $k+1$ *pseudo-balancers*, each with two bits of memory. The first bit describes its *state*: either `up` or `down`, indicating that the next token should be forwarded to the top or bottom output wire, respectively. The second bit describes its *status*: either it is a `leader` or a `follower`. Initially, the first pseudo-balancer is a leader while the others are followers. An inaccessible pseudo-balancer is bypassed; that is, tokens are forwarded directly to the next pseudo-balancer along the same wire that they are received on. Tokens are colored with one of two colors: red indicating that they have been balanced, or green indicating that they have not. Tokens entering a fault-tolerant balancer are initially colored green.

Leader:

A leader balances tokens in the usual fashion. It accepts tokens on either of its input wires, and forwards them alternately to its top and bottom output wires, toggling its state from `up` to `down` or vice-versa. It colors all outgoing tokens red.

Follower:

A follower's behavior differs for red and green tokens. A follower only accepts red tokens in order: first one from its top input wire, then one

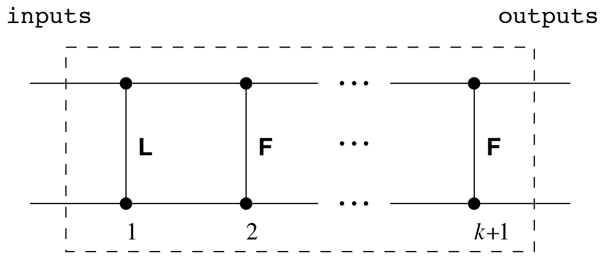

Figure 1 A fault-tolerant balancer (`L` = leader, `F` = follower).

from its bottom input wire, and so on. As it receives red tokens, it toggles its state from `up` to `down`, or vice-versa, and forwards the tokens along the same wire that it receives them on. A green token is an indication that all pseudo-balancers before it have failed. Thus, as soon as a follower receives a green token on either input wire, it becomes a leader and starts routing tokens as described above.

Theorem 12.1 *With at most k faults, the outputs of a k-fault-tolerant balancer are balanced.*

Proof: Omitted. □

Note that there is fine-grained synchronization among processes shepherding tokens concurrently through a fault-tolerant balancer. If a follower receives a red token on the wrong wire (on the bottom wire if its state is `up`, or on the top wire if its state is `down`) then it will block the token. However, if this occurs, then the arrival of a token on the other wire is pending.

4 REMAPPING FAULTY BALANCERS

With some implementations, it may be possible to remap faulty balancers to spare balancers. For shared-memory implementations, this remapping is accomplished by redirecting the pointers of balancers preceding an inaccessible balancer to a spare balancer. The spare balancer is given a random initial state. If this state is different from the original balancer's state, the situation is equivalent to an atomic write operation (by some outside agent) to the memory location holding the balancer's state variable: `up` is change to `down`, or vice-versa.

Denote by x_t and x_b the number of tokens received on a balancer's top and bottom input wires, respectively. Similarly, denote by y_t and y_b the number of tokens forwarded to its top and bottom output wires, respectively. Remapping the balancer alters its outputs as follows:

$$y_t = \left\lceil \frac{x_t + x_b}{2} \right\rceil + f,\ y_b = \left\lfloor \frac{x_t + x_b}{2} \right\rfloor - f$$

for some $f \in \{-1, 0, 1\}$.

Consider a balancer with outputs on wires i and j of some stage of a balancing network. Suppose that the balancer does not experience any faults. Let the output sequence of the stage be $\mathbf{y} = y_0, y_1, \ldots, y_i, \ldots, y_j, \ldots, y_{n-1}$. If instead the balancer fails and is remapped, the output sequence of the stage is $\mathbf{y}' = y_0, y_1, \ldots, y'_i, \ldots, y'_j, \ldots, y_{n-1}$, where $y'_i = y_i + f$ and $y'_j = y_j - f$ for some $f \in \{-1, 0, 1\}$. Clearly, the distance between $\mathbf{y}$ and $\mathbf{y}'$ is less than or equal to 1. With k remapped balancers, the distance between the two sequences is less than or equal to k.

4.1 ERROR BOUND

We will show the following result: remapping k faulty balancers causes an error of at most k in the output token distribution of a balancing network. The following analysis is a generalization of the error bound for sorting networks with faulty comparators by Yao & Yao [10]. Related work can also be found in the paper by Schimmler and Starke [6].

Lemma 12.1 *Balancing the same entries in two sequences cannot increase the distance between them.*

Proof: Omitted. □

Theorem 12.2 *Consider two identical balancing networks given the same input sequence. If there are no faulty balancers in the first and there are k remapped faulty balancers in the second, then the distance between the output sequences of the two networks is less than or equal to k.*

Proof: Omitted. □

4.2 SMOOTHNESS ERROR

We define an error measure on a sequence with respect to the smoothness property.

Definition 12.1 *The* smoothness error *of a sequence is the minimum distance between that sequence and a smooth sequence of the same length with the same total sum.*

A fault-free counting network produces step output sequences. By Theorem 12.2, the distance between the output sequence of a counting network with k remapped faulty balancers and a step sequence with the same total sum is at most k. Since every step sequence is a smooth sequence, the smoothness error of the output sequence of a counting network with k remapped faulty balancers is at most k.

5 TOLERATING FAULTS (2ND METHOD)

Our second method for tolerating faults is to append a correction network to a counting network, as shown in Figure 2. The correction network is constructed with fault-tolerant balancers. Faulty balancers in the counting network are remapped to spare balancers, as described in Section 4. We show the following property for the correction network: given the output token distribution of a counting network with a bounded error, the correction network produces a token distribution that is smooth. In order to tolerate k faults, the correction network consists of $k(\log n+1)$ stages of fault-tolerant balancers, each with $2(k+1)$ bits. Thus, a k-fault-tolerant construction consists of a counting network with depth $O(\log^2 n)$ and a correction network with depth $2k(k+1)(\log n+1)$. Recall that the construction in Section 3 had depth $O(k \log^2 n)$. Therefore, this construction has a smaller depth provided that $O(k) < O(\log n)$. Note, however, that this method is only applicable if it is possible to remap faulty balancers.

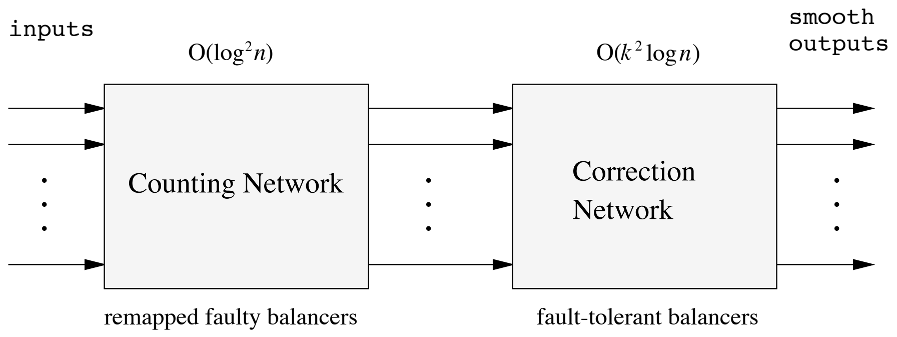

Figure 2 A correction network appended to a counting network.

5.1 CORRECTION NETWORK

In this section, we describe the construction of a balancing network called a correction network with the following property: given an input sequence with a smoothness error of at most k, it produces a smooth output sequence. This network is appended to a counting network in which at most k faulty balancers are remapped. Since the output sequence of the counting network has a smoothness error of less than or equal to k, the final output sequence from the correction network is smooth.

The correction network is constructed from blocks that we call `CORRECT[`n`]` networks. To tolerate k remapped faulty balancers in a counting network of width n, we append k copies of `CORRECT[`n`]`, as shown in Figure 3. Each copy has $\log n + 1$ stages. Note that the `CORRECT[`n`]` networks are built with k-fault-tolerant balancers.

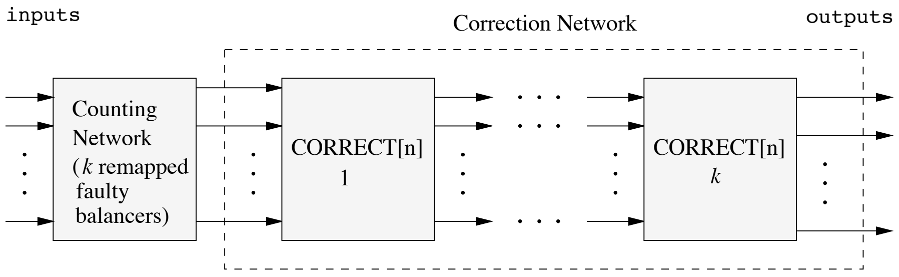

Figure 3 Correcting the output sequence of a counting network with at most k remapped faulty balancers.

Before describing the construction of the correction network, we present the following claim.

Claim 12.1 *Balancing two entries of a sequence cannot increase the smoothness error.*

Proof: Omitted. □

5.1.1 `BUTTERFLY[`n`]` network. In order to construct a `CORRECT[`n`]` network, we require a building block called the `BUTTERFLY[`n`]` network. This network is constructed recursively as follows. For width two, it consists of a single balancer. For width n, where $n = 2^m$ for some $m > 1$, it consists of two `BUTTERFLY[`$n/2$`]` networks with a balancer placed between output wire i of the top network and output wire i of the bottom network, for each $i = 0, \ldots n/2 - 1$ (see Figure 4).

Claim 12.2 *The output value on wire 0 of a* `BUTTERFLY[`n`]` *network is the largest in the output sequence, and the output value on wire* $n-1$ *is the smallest.*

Proof: Omitted. □

Note that it is necessary to balance all the corresponding outputs from the two `BUTTERFLY[`$n/2$`]` networks, in addition to the outputs on wires

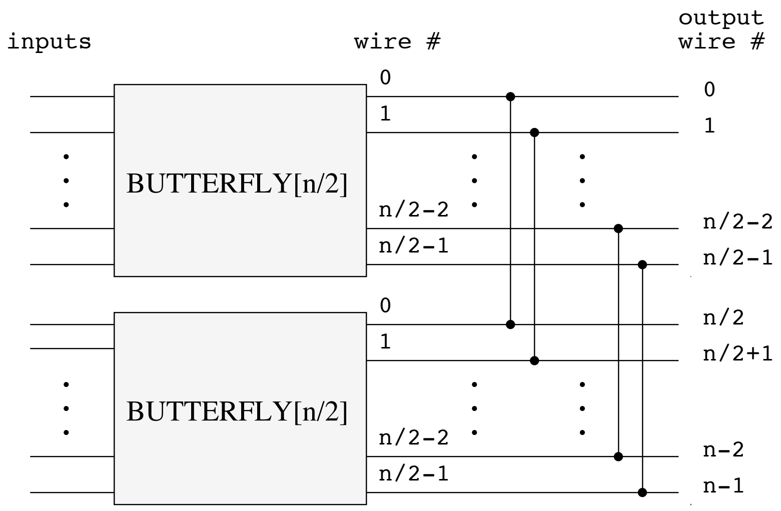

Figure 4 Recursive construction of the `BUTTERFLY[`n`]` network.

0 and $n-1$; otherwise an unbalanced output could be the largest or the smallest in the output sequence.

5.1.2 `CORRECT[`n`]` network. The `CORRECT[`n`]` network is obtained by placing a balancer between wires 0 and $n-1$ at the output of a `BUTTERFLY[`n`]` network, as shown in Figure 5.

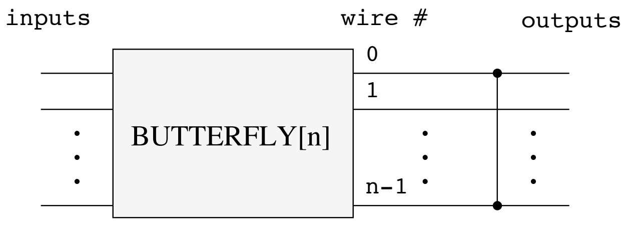

Figure 5 Construction of the `CORRECT[`n`]` network.

Theorem 12.3 *Given an input sequence of length n with a smoothness error of at most k, for some $k \geq 0$, k copies of the* `CORRECT[`n`]` *network produce a smooth output sequence.*

Proof: Omitted. □

6 DISCUSSION

The construction described in Section 5 ensures that the token counts on the output wires differ by at most one. This smoothness property is weaker than the step property of counting networks in the sense that every step sequence is a smooth sequence, but not vice-versa. However, for applications such as load-balancing, a smoothing network is just as effective as a counting network.

We have presented an upper bound on the error resulting from remapping faulty balancers in balancing networks: each remapped faulty balancer causes an error of at most one in the output token distribution. Also, we have presented a practical method for tolerating up to k faults in a counting network, with an increase in the depth of $2k(k+1)(\log n+1)$ for a network of width n. Provided that $O(k^2) < O(\log n)$, this is small compared to the depth of the counting network itself, which is $O(\log^2 n)$ for all practical constructions. Future work is needed to derive lower bounds on the depth of correction networks, and to extend these concepts to *diffracting trees*, a variation of counting networks proposed by Shavit et al. [7][8][9].

References

[1] Y. Afek, M. Merritt and G. Taubenfeld, "Benign Failure Models for Shared-Memory", Lecture Notes in Computer Science, Vol. 725, pp. 69-83, 1993.

[2] Y. Afek, D.S. Greenberg, M. Merritt and G. Taubenfeld, "Computing with Faulty Shared Objects", *Journal of the ACM*, Vol. 42., No. 6, pp. 1231-1274, 1995.

[3] J. Aspnes, M. Herlihy and N. Shavit, "Counting Networks", *Journal of the ACM*, Vol. 41, No. 5, pp. 1020-1048, 1994.

[4] P. Jayanti, T.D. Chandra and S. Toueg, "Fault-Tolerant Wait-Free Shared Objects", *Journal of the ACM*, Vol. 45, No. 3, pp. 451-500, 1998.

[5] A. Orda and M. Merritt, "Efficient Test-and-Set Constructions for Faulty Shared-Memory", *Information Processing Letters*, Vol. 62, No. 1, pp. 41-46, 1997.

[6] M. Schimmler and C. Starke, "A Correction Network for N-Sorters", *SIAM J. Comput.*, Vol. 18, No. 6, pp. 1179-1187, 1989.

[7] N. Shavit and A. Zemach, "Diffracting Trees", *ACM Trans. Computer Systems*, Vol. 14, No. 4, pp. 385-428, 1996.

[8] N. Shavit, E. Upfal, and A. Zemach, "A Steady State Analysis of Diffracting Trees", *Proceedings 8th ACM Symp. Parallel Algorithms and Architectures*, pp. 33-41, 1996.

[9] N. Shavit and D. Touitou, "Elimination Trees and the Construction of Pools and Stacks", *Theory of Computing Systems*, Vol. 30, No. 6, pp. 645-670, 1997.

[10] A. Yao and F. Yao, "On Fault-Tolerant Networks for Sorting", *SIAM J. Computing*, Vol. 14, No. 1, pp. 120-128, 1985.