Network Algorithms

Distributed Counting
Distributed Counting

Counter is a shared variable, supporting atomic Test-and-Increment. The requesting processor gets the value and the counter is then incremented.

Applications? E.g., load-balancing!
Send to server i (mod n)...
Or: renaming identifiers to \{1,\ldots,n\}
How to implement?
Simple Solution

Simple Algo
Send to some node $v$, $v$ replies and increments.

"I have counter!"

Far away, congestion, …:
Not very scalable 😞
Can I count in a decentralized way??
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!

Idea 1: replace comparators with perfect balancers!
Balancer (B)

Values at “in” are sent to upper and lower “out” in turns:

Simply a “toggler”:
Balancer Example

Assume this time of arrival:

\[ \begin{align*}
7 & \quad 6 & \quad 4 & \quad 2 & \quad 1 \\
5 & \quad 3
\end{align*} \]

Order at output?
Balancer Example

Assume this time of arrival:

```
    7 6 4 2 1
  5 3
```

```
   1 3 5 7
  2 4 6
```

The top wire has one more than the lower wire! Coincidence?
Balancer Example

Assume this time of arrival:

```
  7 6 4 2 1
  5 3
```

The top wire has one more than the lower wire! Coincidence?

No, will always be almost perfectly balanced!
Since first goes to top: top may have 1 more in case of an odd number of requests…
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!

Idea 2: add counters at end!
Mini-Counters: distributed!
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!

Idea 3: to count, just send a request through network! (Routed by balancers…)
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!

Idea 3: to count, just send a request through network! (Routed by balancers…)
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!

Idea 3: to count, just send a request through network! (Routed by balancers…)

inc, inc, inc!

mini-counters
A crazy idea?

Batcher networks permute values: use it for balancing the counting?!

Idea 3: to count, just send a request through network! (Routed by balancers…)
The BCN: Overview

Bitonic Counting Network (BCN)

1. Take Batcher and replace comparators with balancers
2. When a node wants to count: sends message to an arbitrary input wire
3. Message routed through the network
4. At outputs make “mini-counter”
5. The mini-counter of wire k replies the value “k+i*w” to the initiator of the ith-message received.
Example

Bitonic Counter Network BCN(4):

2xBCN(2)  2xBSS(2)  M(4)  MN(4)

mini-counters
Example

Bitonic Counter Network BCN(4):

11x Count!

Count!

2xBCN(2)

2xBSS(2)

M(4)

MN(4)
Example

Bitonic Counter Network BCN(4):

So depth / increment time? $\log^2 n$, as for sorting.
Example With Requests

General pattern:

Request propagation?
Example With Requests

General pattern:

Frequency distribution (2,2,2,1): called Step Property!
Formula for requests arriving at i-th counter from top?
Token t mod w = i, every i-th request!
Perfectly load balanced and consistent 😊
**Example With Requests**

**General pattern:**

So what should mini-counter \( i \) return upon its \( r \)-th request? 

\[(r-1)*w + i\]

---

**Frequency distribution \((2,2,2,1)\): called Step Property!**

Formula for requests arriving at \( i \)-th counter from top? 

Token \( t \mod w = i \), every \( i \)-th request! 

Perfectly load balanced and consistent 😊
Analysis

Correctness

In a quiescent state, the w output wires of a bitonic counting network of width w have the step property.

Step Property

A sequence \( y_0, \ldots, y_{w-1} \) has the step property iff:
For any \( i < j \):
\[ 0 \leq y_i - y_j \leq 1 \]

Quiescent State

No message in transit.

Example:
8,8,8,8,7,7,7,7,7

Idea: if output wires have this property, exactly the values 1, ..., r will be assigned by the mini-counters!

Note: Step property implies that values differ by at most 1, and large values come first!
Balancer Properties

Let $x_i$ denote the number of messages consumed on input $i$, where $i = \{0, 1\}$. Denote $y_i$ denote the number of messages on output wire $i$. Then:

(A1) $x_0 + x_1 \geq y_0 + y_1$

(A2) In a quiescent state: $x_0 + x_1 = y_0 + y_1$

(A3) The number of messages sent to the upper output wire is at most one higher than the number on the lower:

$y_0 = \text{ceil}((y_0 + y_1)/2)$, $y_1 = \text{floor}((y_0 + y_1)/2)$

We assume that message is sent to upper first.
Some Facts (1)

Step Property Properties

If a sequence $y_0, y_1, \ldots, y_{w-1}$ has step property,

(B1) All subsequences have step property too

(B2) And even and odd subsequences satisfy:

\[
\sum_{i=0}^{w/2-1} y_{2i} = \left\lfloor \frac{1}{2} \sum_{i=0}^{w-1} y_i \right\rfloor \quad \text{and} \quad \sum_{i=0}^{w/2-1} y_{2i+1} = \left\lceil \frac{1}{2} \sum_{i=0}^{w-1} y_i \right\rceil
\]

Example: $7,7,7,6,6,6,6,6 = 51$

$26 = \text{ceil}(51/2)$

$25 = \text{floor}(51/2)$
More Step Properties

Given two sequences $x_0, x_1, \ldots, x_{w-1}$ and $y_0, y_1, \ldots, y_{w-1}$ with the step property,

(C1) If $\sum x_i = \sum y_i$, then $x_i = y_i$ for any $i$

(C2) If $\sum x_i = \sum y_i + 1$, then there exists a unique $j$ such that $x_j = y_j + 1$, and $x_i = y_i$ for any other $i$.

Example: $x = 7, 7, 7, 7, 6, 6, 6, 6 = 52$
$y = 7, 7, 7, 6, 6, 6, 6, 6, 6 = 51$
Correctness

The step property is maintained inductively!

Proof. Also by induction over balancers, s. lecture notes.

However, only correct in “quiescence state”. But: Counters will always be unique and there are no gaps (gap means request still in transit!), just sometimes a larger counter is issued before a smaller counter!
Linearizability

A system is linearizable if the order in which values are assigned reflects the real-time order in which they were requested: if there is a pair of operations $o_1$, $o_2$, where operation $o_1$ terminates before operation $o_2$ starts, and the logical order is “$o_2$ before $o_1$”, then a distributed system is not linearizable.

Is Bitonic Counting Network linearizable?
Not linearizable!
Not linearizable!

What happens?
Not linearizable!

@1 \rightarrow Zzz

@2 \rightarrow Zzz

@3 \rightarrow 2!
Not linearizable!

What happens now?
Not linearizable!
Not linearizable!

Request started and finished earlier, but got higher number...
The BCN: Overview

**Bitonic Counting Network (BCN)**

1. Take Batcher and replace comparators with balancers
2. When a node wants to count: sends message to an arbitrary input wire
3. Message routed through the network
4. At outputs make “mini-counter”
5. The mini-counter of wire k replies the value “k+i*w” to the initiator of the ith-message received.
**Counting vs Sorting**

Any counting network $C$ is also a sorting network $I(C)$, when replacing balancers with comparators.

**Proof.** Assume: want to sort binary vector $\{0,1\}^*$ in $I(C)$. Make a “request” on each wire of $C$ if there is a “0” at corresponding wire of $I(C)$. Since $C$ is a counting network, all requests end up at upper wires. A comparator of $I(C)$ will receive a “0” on its upper wire iff the corresponding balancer in $C$ receives a request on upper wire (correspondence between requests and “0”); similarly for lower wire. By induction: 0s always exit upper wires (like requests).

The other direction is not true (already Odd/Even network)…
Merger $MR(w)$

Input width $w$

$X \leftarrow X'$

$w/2 \leftarrow w/2$

Balancer

Output $y$

$\cdots$
Recursive definition: $\text{MR}(w)$ from two $\text{MR}(w/2)$

Input $x$

- $x_0$
- $x_{w/2}$
- $x_{w/2-1}$
- $x_{w-1}$

Output $y$

Split odd/even recursive definition pairwise balancers

Upper merges even subsequence, lower merges odd subsequence!
Example: MR(8)

- Even subsequence to top
- Odd subsequence to bottom
- Top merger to upper entry
- Lower merger to lower entry
Backup: Proof

**Merger MR**

If $MR(w)$ has inputs $x_0,x_1,\ldots,x_{w/2-1}$ and $x_w,x_{w+1},\ldots,x_{w-1}$

With step property, then also output $y_0,y_1,\ldots,y_{w-1}$ has step property.
Base case $w=2$: Trivial as $\text{MR}(2)$ is a balancer
Backup: Proof by Induction

Case $w > 2$:

1. Let $z_0, z_{w/2-1}$ and $z'_0, z'_{w/2-1}$ be output of upper and lower $\text{MR}(w/2)$
2. Even & odd sub-sequences $x_0, x_1, ..., x_{w/2-1}$ and $x_w, x_{w+1}, ..., x_{w-1}$ resp. must have step property too, see (B1)
3. And by Induction Hypothesis, $z_0, z_{w/2-1}$ and $z'_0, z'_{w/2-1}$ have step property
4. Let $Z = \sum z_i$ and $Z' = \sum z'_i$, so by (B2) even and odd subsequences satisfy:

$$Z = \left[ \frac{1}{2} \sum_{i=0}^{w/2-1} x_i \right] + \left[ \frac{1}{2} \sum_{i=w/2}^{w-1} x_i \right] \quad Z' = \left[ \frac{1}{2} \sum_{i=0}^{w/2-1} x_i \right] + \left[ \frac{1}{2} \sum_{i=w/2}^{w-1} x_i \right]$$
5. So Z and Z’ differ by at most 1.
   5.a. If Z and Z’ are the same, (C1) implies that \( z_i = z_i' \), so output has step property too.
   5.b. If Z and Z’ differ by one, (C2) implies that there exists a unique j where \( z_j \) and \( z_j' \) differ. Let \( l := \min(z_j, z_j') \), so output \( y_i = l+1 \) for \( i < 2j \) and \( y_i = l \) for \( i > 2j+1 \). The final balancer ensures that \( y_{2j} = l+1 \) and \( y_{2j+1} = l \), so also step property.