Concurrent Counting is harder than Queuing

Costas Busch
Rensselaer Polytechnic Institute

Srikanta Tirthapura
Iowa State University
Arbitrary graph
Distributed Counting

Some processors request a counter value
Distributed Counting

Final state
Distributed Queuing

Some processors perform enqueue operations
Distributed Queuing

Previous=D

Previous=nil

Previous=A

Previous=B

Previous=B

C A D B

Tail

head
Applications

Counting:
- parallel scientific applications
- load balancing (counting networks)

Queuing:
- distributed directories for mobile objects
- distributed mutual exclusion
Ordered Multicast

Multicast with the condition:  
all messages received at all nodes in  
the same order

Either Queuing or Counting will do

Which is more efficient?
Queuing vs Counting?

Total orders

Queuing = finding predecessor
Needs local knowledge

Counting = finding rank
Needs global knowledge
Problem

Is there a formal sense in which Counting is harder problem than Queuing?

Reductions don’t seem to help
Our Result

Concurrent Counting is harder than Concurrent Queuing

on a variety of graphs including:
many common interconnection topologies
  complete graph,
  mesh
  hypercube
  perfect binary trees
Model

Synchronous system $G=(V,E)$
- edges of unit delay

Congestion: Each node can process only one message in a single time step

Concurrent one-shot scenario:
- A set $R$ subset $V$ of nodes issue queuing (or counting) operations at time zero
- No more operations added later
Cost Model

$C_Q(v)$: delay till $v$ gets back queuing result

Cost of algorithm $A$ on request set $R$ is

$$C_Q(A, R) = \sum_{v \in R} C_Q(v)$$

Queuing Complexity = $\min_A \{\max_{R \subseteq V} C_Q(A, R)\}$

Define Counting Complexity Similarly
Lower Bounds on Counting

For arbitrary graphs:

\[
\text{Counting Cost} = \Omega(n \log^* n)
\]

For graphs with diameter \( D \):

\[
\text{Counting Cost} = \Omega(D^2)
\]
Theorem: For graphs with diameter $D$:

Counting Cost $= \Omega(D^2)$

Proof:

Consider some arbitrary algorithm for counting
Take shortest path of length $D$.
Graph

make these nodes to count
Node of count $k$ decides after at least $\frac{k-1}{2}$ time steps

Needs to be aware of $k-1$ other processors
Counting Cost: \[ \sum_{k=1}^{D} \frac{k - 1}{2} = \Omega(D^2) \]

End of Proof
Theorem: For arbitrary graphs:

\[ \text{Counting Cost} = \Omega(n \log^* n) \]

Proof:

Consider some arbitrary algorithm for counting
Prove it for a complete graph with \( n \) nodes: any algorithm on any graph with \( n \) nodes can be simulated on the complete graph.
The initial state affects the outcome

Red: count
Blue: don’t count
Red: count
Blue: don’t count

Final state
Red: count
Blue: don’t count

Initial state
Red: count
Blue: don’t count

Final state
Let $A(v)$ be the set of nodes whose input may affect the decision of $V$. 
Suppose that there is an initial state for which \( V \) decides \( k \)

Then: \(|A(v)| \geq k\)
These two initial states give same result for $\nu$
If $|A(v)| < k$, then $v$ would decide less than $k$

Thus, $|A(v)| \geq k$
Suppose that $v$ decides at time $t$

We show:

$$|A(v)| \leq 2^2 \cdot \ldots \cdot 2 \text{ times}$$
Suppose that \( v \) decides at time \( t \)

\[
|A(v)| \leq 2^2 \quad 2 \ldots 2 \quad t \text{ times} \quad t \geq \log^* k
\]

\[
|A(v)| \geq k
\]
Cost of node $V: \quad t \geq \log^* k$

If $n$ nodes wish to count:

Counting Cost = $\sum_{k=1}^{n} \log^* k = \Omega(n \log^* n)$
Nodes that affect $v$ up to time $t$

$$a(t) = \max_x |A(x,t)|$$

Nodes that $v$ affects up to time $t$

$$b(t) = \max_x |B(x,t)|$$
After $t = 1$, the sets grow
\[ A(v, t + 1) \]

\[ A(v, t) \]

\[ v \]
There is an initial state such that \( z \) sends a message to \( v \).
Suppose that $A(s,t) \cap A(z,t) = \emptyset$

Then, there is an initial state such that both send message to $v$
However, \( v \) can receive one message at a time.
Therefore: \[ A(s,t) \cap A(z,t) \neq \emptyset \]
Number of nodes like $s$:

$$a(t) \cdot b(t)$$

$$\max_x |A(x,t)|$$

$$\max_x |B(x,t)|$$
Therefore: \[|A(v, t + 1)| \leq |A(v, t)| + a(t) \cdot (a(t) \cdot b(t))\]
Thus: \[ a(t + 1) \leq a(t)(1 + a(t)b(t)) \]

We can also show: \[ b(t + 1) \leq b(t)(1 + 2^{a(t)}) \]

Which give:

\[ a(\tau) \leq 2^2 \times 2^{\tau \text{ times}} \]

End of Proof
Upper Bound on Queuing

For graphs with spanning trees of constant degree:

$\text{Queuing Cost} = O(n \log n)$

For graphs whose spanning trees are lists or perfect binary trees:

$\text{Queuing Cost} = O(n)$
An arbitrary graph
Spanning tree
Spanning tree
Distributed Queue

Previous = Nil
enq(B)

Previous = Nil

Tail

A

Previous = ?

B

Head

Tail

A
A informs B
Concurrent Enqueue Requests

enq(C) C

Previous = ?

B enq(B)

Previous = ?

A

Previous = Nil

Tail

A

Tail

Head
enq(C)

Previous = Nil

Tail

enq(B)

Previous = ?

Tail

Head
Previous = A

Previous = Nil

enq(B)

Previous = ?
enq(B) C

Previous = A

Previous = Nil

Tail

B

Previous = ?

A

Tail

Head

C

A
C informs B

Previous = A

Previous = Nil

Tail

Head
Paths of enqueue requests

Previous = A
Previous = Nil
Previous = C
Nearest-Neighbor TSP tour on Spanning tree

Origin (first element in queue)
Visit closest unused node in Tree
Visit closest unused node in Tree
Nearest-Neighbor TSP tour
For spanning tree of constant degree:

\[
\text{Queuing Cost} \leq 2 \times \text{Nearest-Neighbor TSP length}
\]

[Herlihy, Tirthapura, Wattenhofer PODC’01]
If a weighted graph satisfies triangular inequality:

\[
\text{Nearest-Neighbor TSP length} \leq \text{Optimal TSP length} \times \log n
\]
weighted graph of distances
Satisfies triangular inequality

\[ \quad w(e_1) \leq w(e_2) + w(e_3) \]
Nearest Neighbor TSP tour

Length=8
Optimal TSP tour

Length = 6
It can be shown that:

\[ \text{Optimal TSP length} \leq 2n \] (Nodes in graph)

Since every edge is visited twice
Therefore, for constant degree spanning tree:

**Queuing Cost** = \( O(\text{Nearest-Neighbor TSP}) \)

= \( O(\text{Optimal TSP} \times \log n) \)

= \( O(n \log n) \)
For special cases we can do better:

Spanning Tree is

List

balanced binary tree

Queuing Cost = $O(n)$
Graphs with Hamiltonian path, have spanning trees which are lists.

Complete graph

Mesh

Hypercube

Queuing Cost = $O(n)$

Counting Cost = $\Omega(n \log^* n)$
Theorem:

If the spanning tree is a list, then

Queuing Cost = $O(n)$
Proof:

Nearest Neighbor TSP

Queuing Cost = \( O(\text{Nearest-Neighbor TSP}) \)
$x < y$

$x$

$y$

$a$

$b$

$b > 2a$
$n$ nodes

Even sides

Length doubles

Total length $\leq 2n$
$n$ nodes

Odd sides

Length doubles

Total length $\leq 2n$
\( n \) nodes

Total length \( \leq 2n + 2n = 4n \)

End of proof