

# On the Stability of Dynamic Diffusion Load Balancing

Russell Martin

University of Liverpool

March 4, 2008



Joint work with

**Petra Berenbrink**  
(Simon Fraser University)

**Tom Friedetzky**  
(University of Durham)

# The General Framework

We consider the problem of *dynamic load balancing*.

# The General Framework

We consider the problem of *dynamic load balancing*.

- Processors are vertices in a connected graph.
- Edges in the graph denote communications linkages where tasks may be sent from one processor to another.

# The General Framework

We consider the problem of *dynamic load balancing*.

- Processors are vertices in a connected graph.
- Edges in the graph denote communications linkages where tasks may be sent from one processor to another.
- In each time step,
  - some new tasks are inserted into the system on the processors;
  - load is balanced amongst processors according to some predefined method;
  - finally, each non-empty processor deletes one task.

# What do we want to know?

Under what conditions is this process *stable*?

# What do we want to know?

Under what conditions is this process *stable*?

*Stability* means that the total system load is bounded as a function of  $n$  (= number of processors) alone, independent of time.

# What do we want to know?

Under what conditions is this process *stable*?

*Stability* means that the total system load is bounded as a function of  $n$  (= number of processors) alone, independent of time.

Obviously, we can insert (on average) at most  $n$  tasks into the system during any time step.

# Previous Work

Lots of work has been done in the *static* setting (i.e. no new tasks generated, none deleted).

# Previous Work

Lots of work has been done in the *static* setting (i.e. no new tasks generated, none deleted).

Some of this work operates under various restrictions like an edge can only forward one task per round, or considers first- and second-order schemes, etc.

## Dynamic load balancing

**Muthukrishnan and Rajaraman** [1998]

Edges can forward a single task, adversarial load generation, but limited by edge cuts (at most  $(1 - \epsilon)e(S, \bar{S})$  tasks inserted/deleted in a set  $S$ ).

**Berenbrink, Friedetzky, and Mayr** [1998]

Use a “collision protocol” to resolve load balancing requests between processors, randomized task generation where each processor receives (in expectation) at most one new task per round.

**Anshelevich, Kempe, and Kleinberg** [2002]

Positive result in the setting of Muthukrishnan and Rajaraman for  $\varepsilon = 0$ , when edges that can pass a constant number of tasks per round.

**Anagnostopoulos, Kirch, and Upfal** [2003]

Balancing procedure involves choosing a random matching at each step so a processor is involved in only a single balancing action each round. Insert at most  $\lambda tn$  tasks over an interval of time  $t$ , where  $\lambda < 1$ . Not extendable to the case  $\lambda = 1$ .

## Work Stealing

Empty processors receive (or request) load from other processors. (No other load balancing is performed.)

**Berenbrink, Friedetzky, Goldberg** [2003]

$n$  generators that each generate a task with probability strictly less than 1. Processors form a *complete graph*. Showed stability under a wide range of conditions.

# Our Setting

$G$  is a connected graph on  $n$  vertices with max degree  $\Delta$ .

# Our Setting

$G$  is a connected graph on  $n$  vertices with max degree  $\Delta$ .

During each time step:

- We insert  $n$  tasks (randomly or deterministically) into the system.

# Our Setting

$G$  is a connected graph on  $n$  vertices with max degree  $\Delta$ .

During each time step:

- We insert  $n$  tasks (randomly or deterministically) into the system.
- Load is balanced according to the following procedure, for every pair of processors (simultaneously):  
Processor  $i$  sends

$$\max \left\{ 0, \left\lfloor \frac{\ell_i - \ell_j}{2 \max\{d_i, d_j\}} \right\rfloor \right\}$$

tasks to processor  $j$ .

# Our Setting

$G$  is a connected graph on  $n$  vertices with max degree  $\Delta$ .

During each time step:

- We insert  $n$  tasks (randomly or deterministically) into the system.
- Load is balanced according to the following procedure, for every pair of processors (simultaneously):  
Processor  $i$  sends

$$\max \left\{ 0, \left\lfloor \frac{\ell_i - \ell_j}{2 \max\{d_i, d_j\}} \right\rfloor \right\}$$

tasks to processor  $j$ .

- Each non-empty processor deletes, or consumes, one task.

The “usual” load balancing function that has been considered in this setting is

$$\max \left\{ 0, \left\lfloor \frac{l_i - l_j}{\max\{d_i, d_j\} + 1} \right\rfloor \right\}.$$

The difference is an artifact of our proof.

# Features of this method

- No conditions on  $G$  are necessary, other than connectivity.

# Features of this method

- No conditions on  $G$  are necessary, other than connectivity.
- No global knowledge is needed. Each vertex needs information only about its immediate neighborhood.

# Features of this method

- No conditions on  $G$  are necessary, other than connectivity.
- No global knowledge is needed. Each vertex needs information only about its immediate neighborhood.
- We achieve *task saturation*, i.e.  $n$  tasks are inserted into the system in each time step.

## Theorem

*Let  $G$  be a connected graph on  $n$  vertices with max degree  $\Delta$ .*

*The load balancing procedure previously outlined is stable.*

*Starting from an empty system, the maximum load at the end of any time step is at most  $2\Delta n^2(n + 1)$ .*

# Proof idea

We use a similar function to that used by Anshelevich, Kempe, and Kleinberg [2002] (though the similarities end there).

For any subset,  $S$ , of processors, we show that the total load of  $S$  at the end of any round satisfies:

$$L(S) \leq \sum_{k=n-|S|+1}^n k \cdot (4\Delta) \cdot n.$$

This immediately implies the theorem.

# Proof idea

We use a similar function to that used by Anshelevich, Kempe, and Kleinberg [2002] (though the similarities end there).

For any subset,  $S$ , of processors, we show that the total load of  $S$  at the end of any round satisfies:

$$L(S) \leq \sum_{k=n-|S|+1}^n k \cdot (4\Delta) \cdot n.$$

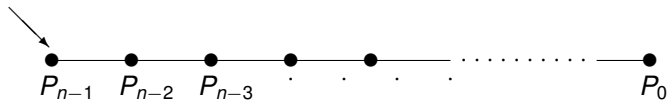
This immediately implies the theorem.

The general idea of the proof is inductive in nature, i.e. assume the condition holds at time  $t$  and show it holds at time  $t + 1$ .

(Further details are omitted...)

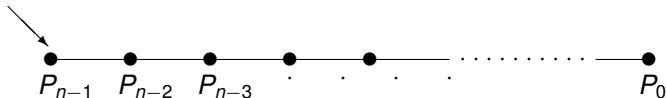
# A Lower Bound for the Path

Insert  $n$  tasks each round.



# A Lower Bound for the Path

Insert  $n$  tasks each round.



After the initial “burn-in” rounds where load is distributed from left to right, the load of  $P_k$  at the end of a round is  $2k(k + 1)$ .

The total system load is

$$\sum_{k=0}^{n-1} 2k(k + 1) = \frac{2}{3}n(n^2 - 1).$$

# Hold on a second!

For  $K_n$  our result implies an upper bound of  $O(n^4)$  for the total load in the system, which surely can't be right.

# Hold on a second!

For  $K_n$  our result implies an upper bound of  $O(n^4)$  for the total load in the system, which surely can't be right.

## Open Problem

Does the upper bound on the total load in the system *really* depend upon the maximum degree or, say, the average degree?

What is the truth?

# Work stealing doesn't cut it (in general)

Suppose we consider the work stealing scenario, i.e. only empty processors request (or “steal”) load from their non-empty neighbors.

In this case, we can consider a load balancing function like

$$\alpha_{i,j} = \begin{cases} \lfloor \frac{\ell_j}{d_{i+1}} \rfloor & \ell_j = 0 \text{ and } j \sim i \\ 0 & \text{otherwise} \end{cases}$$

i.e. processor  $j$  (that is empty) takes load from its non-empty neighbors.

## Work stealing (cont.)

Berenbrink, Friedetzky, and Goldberg proved their positive results on work stealing under the condition that *the underlying graph is a complete graph*.

## Work stealing (cont.)

Berenbrink, Friedetzky, and Goldberg proved their positive results on work stealing under the condition that *the underlying graph is a complete graph*.

As soon as we have a pair of vertices  $u$  and  $v$  for which there is no edge joining them, then (given enough tasks) work stealing isn't enough to ensure stability.

## Work stealing (cont.)

Berenbrink, Friedetzky, and Goldberg proved their positive results on work stealing under the condition that *the underlying graph is a complete graph*.

As soon as we have a pair of vertices  $u$  and  $v$  for which there is no edge joining them, then (given enough tasks) work stealing isn't enough to ensure stability.

An adversary can allocate one task on all  $d_u$  neighbors of  $u$ , and two tasks on  $u$ . (This is still at most  $n$  tasks injected into the system.)

Then  $u$  will never balance load with its neighbors, and hence the load of  $u$  strictly increases in each round.

That's all folks...

Thanks!