This talk covers joint work done with

Petra Berenbrink \textsuperscript{1}
Tom Friedetzky \textsuperscript{2}
Leslie Ann Goldberg \textsuperscript{3}
Paul Goldberg \textsuperscript{3}
Zengjian Hu \textsuperscript{1}

\textsuperscript{1} Simon Fraser University
\textsuperscript{2} University of Durham
\textsuperscript{3} University of Liverpool
Set of $m$ identical tasks to distribute over a collection of $n$ identical resources.

Obvious solution... \[\lfloor \frac{m}{n} \rfloor\] or \[\lceil \frac{m}{n} \rceil\] tasks on each, and centralized control lets us achieve this easily.
Set of $m$ identical tasks to distribute over a collection of $n$ identical resources.

Obvious solution... $\lfloor \frac{m}{n} \rfloor$ or $\lceil \frac{m}{n} \rceil$ tasks on each, and centralized control lets us achieve this easily.

Alternatively, we consider the problem where each task is actually an “agent” who seeks to minimize their own current load.

Tasks (agents) move concurrently, attempting to decrease their personal load.
Set of $m$ identical tasks to distribute over a collection of $n$ identical resources.

Obvious solution... $\lfloor \frac{m}{n} \rfloor$ or $\lceil \frac{m}{n} \rceil$ tasks on each, and centralized control lets us achieve this easily.

Alternatively, we consider the problem where each task is actually an “agent” who seeks to minimize their own current load.

Tasks (agents) move concurrently, attempting to decrease their personal load.

The agents thus seek a *Nash equilibrium* where none can decrease their own personal load by moving to a different resource.
E. Even-Dar, A. Kesselman, and Y. Mansour [2003]
Weighted tasks, variable capacity resources, tasks use “best response” moves for tasks, tasks move one at a time.

P. Goldberg [2004]
Tasks select resources at random and migrate if they improve, but migrations occur one at a time.

E. Even-Dar and Y. Mansour [2005]
Tasks can move concurrently and independently. Terminates in (expected) $O(\log \log m + \log n)$ rounds. Requires global knowledge of average load, and tasks on underloaded resources don’t move.
A Natural(?) Solution

Let $X_1(t), X_2(t), \ldots, X_n(t)$ denote the current arrangement of tasks.

We consider the following parallel, synchronous protocol:

For each task $b$ do (in parallel)
   Let $i_b$ be the current resource of task $b$.
   Choose a resource $k_b$ uniformly at random.

   If $X_{i_b}(t) > X_{k_b}(t)$ then
      Move task $b$ from resource $i_b$ to $k_b$ with probability $1 - X_{k_b}(t)/X_{i_b}(t)$.
Let $X_1(t), X_2(t), \ldots, X_n(t)$ denote the current arrangement of tasks.

We consider the following parallel, synchronous protocol:

For each task $b$ do (in parallel)
- Let $i_b$ be the current resource of task $b$.
- Choose a resource $k_b$ uniformly at random.
- If $X_{i_b}(t) > X_{k_b}(t)$ then
  - Move task $b$ from resource $i_b$ to $k_b$ with probability $1 - X_{k_b}(t)/X_{i_b}(t)$.

How many steps until we reach equilibrium?
Advantages of this approach

- No global knowledge needed, except for the value of $n$ (the number of processors).
- Distributed, highly parallel, robust when new tasks are added to system.
- Constant amount of work performed by each task (one query).
- The protocol gives each resource the “right amount” (in expectation), thereby giving the agents a (somewhat) compelling reason to use this protocol.
Suppose (w.l.o.g.) that $X_1(t) \geq X_2(t) \geq \cdots \geq X_n(t)$. Then

\[
\mathbb{E}[X_i(t+1)] = X_i(t) + \sum_{k=1}^{i-1} X_k(t) \cdot \frac{1}{n} \left( 1 - \frac{X_i(t)}{X_k(t)} \right)
\]

\[
- \sum_{k=i+1}^{n} X_i(t) \cdot \frac{1}{n} \left( 1 - \frac{X_k(t)}{X_i(t)} \right)
\]

\[
= X_i(t) + \frac{1}{n} \left( \sum_{k=1}^{i-1} (X_k(t) - X_i(t)) \right)
\]

\[
- \sum_{k=i+1}^{n} (X_i(t) - X_k(t))
\]

\[
= \ldots
\]

\[
= \frac{m}{n}
\]
A word of warning...
A word of warning...

Proceed with caution!

This protocol doesn’t really work!
Let $n = m$ and suppose the initial assignment of tasks is $X(0) = (n, 0, 0, \ldots, 0)$.

Let $T$ denote the time when the process reaches equilibrium (i.e. one task on each resource).

Then

$$\mathbb{E}[T] \geq \exp(\Theta(\sqrt{n})).$$
Proof.

(Idea)

Given an assignment $x$, let $n_0(x)$ denote the number of *empty* resources.

Let $\alpha = \lfloor \sqrt{n} \rfloor$. 
Proof.

(Idea)

Given an assignment $x$, let $n_0(x)$ denote the number of empty resources.

Let $\alpha = \lfloor \sqrt{n} \rfloor$.

We show that for any assignment $x$ with $n_0(x) \geq \alpha$, we have

$$\Pr(n_0(X(t+1)) < \alpha \mid X(t) = x) \leq \exp(-\Theta(\sqrt{n})).$$

(Two cases, use Chernoff bounds, and consider “loners” in one of the cases.)
Tasks can move when they really shouldn’t.

For each task $b$ do (in parallel)
  Let $i_b$ be the current resource of task $b$. 
  Choose a resource $k_b$ uniformly at random.

  If $X_{i_b}(t) > X_{k_b}(t)$ then
    Move task $b$ from resource $i_b$ to $k_b$ with
    probability $1 - X_{k_b}(t)/X_{i_b}(t)$.

With “neutral” moves allowed, the process doesn’t terminate (i.e. stay at equilibrium) when $n \not| m$. 
For each task $b$ do (in parallel)

Let $i_b$ be the current resource of task $b$.
Choose a resource $k_b$ uniformly at random.

If $X_{i_b}(t) > X_{k_b}(t) + 1$ then
Move task $b$ from resource $i_b$ to $k_b$ with probability $1 - X_{k_b}(t)/X_{i_b}(t)$.

This natural fix for the second problem actually fixes the other difficulty too!
The “Big Fix”

For each task $b$ do (in parallel)
Let $i_b$ be the current resource of task $b$. Choose a resource $k_b$ uniformly at random.

If $X_{i_b}(t) > X_{k_b}(t) + 1$ then
Move task $b$ from resource $i_b$ to $k_b$ with probability $1 - X_{k_b}(t)/X_{i_b}(t)$.

This natural fix for the second problem actually fixes the other difficulty too!
Success!

This new protocol works!!

**Theorem**

Let $T$ be the number of rounds taken by the new process (without “neutral moves”) to reach equilibrium.

Then

$$\mathbb{E}(T) \in O(\log \log m + n^4).$$
Proof.

(Idea)

For an assignment $X$, define $\Phi(X) = \sum_{i=1}^{n} (X_i - \frac{m}{n})^2$.

(If $n|m$, then $\Phi(X(t)) = 0$ iff $X(t)$ is at equilibrium.
In general, if $r = m \mod n$, then $\Phi(X(t)) \geq r(1 - r/n)$ with equality iff $X(t)$ is at equilibrium.)

We can show, with high probability, after $t \in O(\log \log m)$ steps, we have that $\Phi(X(t)) \in O(n)$.
Proof.

(Idea)

For an assignment $X$, define $\Phi(X) = \sum_{i=1}^{n} (X_i - \frac{m}{n})^2$.

(If $n|m$, then $\Phi(X(t)) = 0$ iff $X(t)$ is at equilibrium. In general, if $r = m \mod n$, then $\Phi(X(t)) \geq r(1 - r/n)$ with equality iff $X(t)$ is at equilibrium.)

We can show, with high probability, after $t \in O(\log \log m)$ steps, we have that $\Phi(X(t)) \in O(n)$.

Then, we can show that $\Phi$ is a supermartingale, i.e.

$$\mathbb{E}[\Phi(X(t+1))] \leq \Phi(X(t)).$$

Show the variance of $\Phi$ is “sufficiently large”, and use martingale convergence theorems to finish off the last bit.
Theorem

Let \( m \) (the number of tasks) be even. For either protocol (with or without “neutral moves”), and the starting configuration \( X(0) = (m, 0) \) (i.e. two resources), we have that

\[
\mathbb{E}[T] \in \Omega(\log \log m)
\]

where \( T \) is the time that the system reaches equilibrium.
Robust, parallel, distributed algorithm (with low overhead) that will balance load in expected $O(\log \log m + n^4)$ time.

Not much global knowledge required.

What is the correct order for the “last bit”? Is it really $O(n^4)$?

Extend to weighted tasks? Non-identical resources?
Now we consider the more traditional approach in load balancing where the **resources** control the balancing procedure.

Each round the resources (or processors) share load amongst themselves according to some protocol (e.g. empty processors request load from others, processors send a fraction of work to their neighbors that are less loaded, etc.).

Also, new tasks are inserted in each round, and tasks are deleted (consumed, processed, etc).
The General Framework

- 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 vertices; load is balanced amongst processors according to some predefined method; finally, each non-empty processor deletes one task. 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. Obviously we can insert (on average) at most $n$ tasks into the system during any time step.
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 vertices;
- load is balanced amongst processors according to some predefined method;
- finally, each non-empty processor deletes one task.
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 vertices;
- load is balanced amongst processors according to some predefined method;
- finally, each non-empty processor deletes one task.

Under what conditions is this process stable?

Stability means that the total system load is bounded as a function of $n$ (i.e., number of processors) alone.
The General Framework

- 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 vertices;
  - load is balanced amongst processors according to some predefined method;
  - finally, each non-empty processor deletes one task.

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.

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

Lots of work in the static setting (i.e. no new tasks generated, none deleted), under various restrictions like an edge can only forward one task per round, 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 $(1 - \varepsilon) e(S, \overline{S})$ 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 $\varepsilon = 0$, and for 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, insert at most $\lambda tn$ tasks over an interval of time $t$, where $\lambda < 1$. Not extendable to the case $\lambda = 1$. 
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 one task.
No conditions other than connectivity are necessary on \( G \).

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.
The Payoff

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

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

This immediately implies the result.

(Details are omitted.)
The Payoff

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.)
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 result.

(Details are omitted.)
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

$$
\sum_{k=0}^{n-1} 2k(k + 1) = \frac{2}{3} n(n^2 - 1).
$$
What about the complete graph?

Our theorem gives an upper bound of $O(n^4)$ for the complete graph $K_n$.

There’s an easy lower bound of $O(n^2)$. Simply consider the processors in a round robin fashion, inserting $n$ tasks on each one. At the end of $n$ rounds, this gives the lower bound stated.
What about the complete graph?

Our theorem gives an upper bound of $O(n^4)$ for the complete graph $K_n$.

There’s an easy lower bound of $O(n^2)$.
Simply consider the processors in a round robin fashion, inserting $n$ tasks on each one. At the end of $n$ rounds, this gives the lower bound stated.

What’s the real answer?
What about the complete graph?

Our theorem gives an upper bound of $O(n^4)$ for the complete graph $K_n$.

There’s an easy lower bound of $O(n^2)$.
Simply consider the processors in a round robin fashion, inserting $n$ tasks on each one. At the end of $n$ rounds, this gives the lower bound stated.

What’s the real answer?
Is it true that the upper bound is really something like $O(n^3/\Delta)$?
We now consider the distribution of tasks, i.e. the usual setting of balls-into-bins games where we have to sequentially distribute tasks (balls) randomly to processors (bins).

A classical result tells us that if we throw $n$ uniform balls into $n$ bins, then the maximum load of any bin is $\Theta(\log n / \log \log n)$ (with high probability).
With more choices, the load can be decreased dramatically.

Azar, Broder, Karlin, and Upfal (1999) showed that if each ball picks two bins (u.a.r.) and places the ball in the least loaded of the bins, the maximum load drops to $\Theta(\log \log n)$ w.h.p.

(In general, with $d > 1$ choices, the maximum load is $\Theta(\log \log n / \log d)$.)
What kind of results can we obtain in the case of weighted balls?

What effect does the distribution of weights have on the maximum load?

What about the multiple-choice setting?

How does the order in which they are distributed affect the maximum load?
Previous Work

Single-choice games (uniform weights)
  • Sanders [1996]
  • Mitzenmacher, Richa, Sitaraman [2000]

Single-choice games (non-uniform weights)
  • Berenbrink, auf der Heide, Schröder [1999]
  • Koutsoupias, Mavronicolas, Spirakis [2003]

Multiple-choice games (uniform weights)
  • Azar, Broder, Karlin, Upfal [1999]
  • Berenbrink, Czumaj, Steger, Vöcking [2000]
  • Mitzenmacher, Richa, Sitaraman [2000]
  • Mitzenmacher, Prabhakar, Shah [2002]

Multiple-choice games (non-uniform weights)
  • Berenbrink, auf der Heide, Schröder [1999]
Single-choice games

The order in which the balls are distributed is not relevant.

An important concept in this case is majorization of vectors.
The order in which the balls are distributed is not relevant.

An important concept in this case is *majorization* of vectors. Let \( u = (u_1, \ldots, u_m) \) and \( v = (v_1, \ldots, v_m) \) be two vectors such that

\[
\begin{align*}
    u_1 &\geq u_2 \geq \cdots \geq u_m \\
v_1 &\geq v_2 \geq \cdots \geq v_m \\
    \sum u_i &= \sum v_i = W.
\end{align*}
\]

Then \( u \) is said to majorize \( v \), written \( u \succ v \), if

\[
\sum_{i=1}^{k} u_i \geq \sum_{i=1}^{k} v_i \quad \text{for all } k = 1, \ldots, m.
\]
Fix a probability distribution $\mathbb{P}$ over $\{1, \ldots, n\}$, the set of bins.

**Theorem**

*Suppose that $u \succ v$.*

*Let $u$ and $v$ be allocated according to the distribution $\mathbb{P}$. Then*

$$\mathbb{E}[\max \text{ load for } u] \geq \mathbb{E}[\max \text{ load for } v].$$

The same result extends when we consider the largest pair, triple, ... of loaded bins in each load vector, i.e. the majorization relation is preserved (in expectation).
Here we use the “greedy” protocol mentioned earlier. Namely, each ball picks two (or, generally, $d$) bins uniformly at random and goes into the least loaded bin.
Here we use the “greedy” protocol mentioned earlier. Namely, each ball picks two (or, generally, $d$) bins uniformly at random and goes into the least loaded bin.

Contrary to the single-choice games, uniform balls do not necessarily minimize the max load.

<table>
<thead>
<tr>
<th>Allocations</th>
<th>First $m/2$ balls</th>
<th>Last $m/2$ balls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>weight</td>
<td>weight</td>
</tr>
<tr>
<td>$A$</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$B$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Theorem**

*After allocating the two sets of balls, we have*

$$\mathbb{E}[\text{max of system } B] \geq \mathbb{E}[\text{max of system } A] + \frac{\log \log n}{\log d} - \Theta(1).$$
Allocating a large number of small balls is not necessarily better than a small number of larger balls of the same total weight (also contrary to the single-choice case).
Allocating a large number of small balls is not necessarily better than a small number of larger balls of the same total weight (also contrary to the single-choice case).

**Theorem**

Consider the two sets of weights

\[ w_C = \langle 1, 1 \rangle \quad \text{and} \quad w_D = \langle \frac{1}{2}, \frac{1}{2}, 1 \rangle, \]

where we allocate the balls in the order shown into \( n \geq 3 \) bins. Then

\[ \mathbb{E}[\text{max of system } C] = 1 + \frac{1}{n^2} \]

\[ \mathbb{E}[\text{max of system } D] = 1 + \frac{2}{n^2} - \frac{1}{n^4} \]
We conjecture the following:

Conjecture

When \( m \geq n \), if the weights are allocated in decreasing order, then the expected maximum load is minimized (amongst all possible permutations).
This conjecture is incorrect.
This conjecture is incorrect.

Consider the two sets of weights (listed in the order of allocation):

\[ A \quad 9, 6, 6, \ldots, 6, 5, 4, 4 \]
\[ B \quad 9, 6, 6, \ldots, 6, 4, 5, 4 \]

Suppose we insert these sets of balls into two bins. Then \( E[\max \text{ load of system } A] > E[\max \text{ load of system } B] \).