Decentralized scheduling: a new analysis of work stealing

Nicolas Gast

EPFL LIG and INRIA

November 17, 2010

joint work with Bruno Gaujat, Marc Tchiboukdjian and Denis Trystram.
Efficient use of today’s platforms

- Parallel platforms
  - Multicores
  - Manycores (GPU)

- Characteristics
  - Increasing number of cores
  - Shared memory

- Parallel Programming:
  Need for parallel algorithms that are
  - efficient (even on small data sets)
  - easy to program
The new standard for parallel programming?

- Cilk, Intel TBB, Microsoft TPL, KAAPI, …
- The programmer declares tasks and dependencies
- Tasks are created at runtime
- The runtime is in charge of the scheduling

\[
\text{FIB}(n) \{
\begin{align*}
\text{if} \quad (n \leq 1) & \quad \text{return } n ; \\
\text{else} & \\
& \quad x = \text{spawn } \text{FIB}(n-1) ; \\
& \quad y = \text{FIB}(n-2) ; \\
& \quad \text{sync} ; \\
& \quad \text{return } x+y ; \\
\end{align*}
\}
\]

Compute Fibonacci number with Cilk

\[
\begin{align*}
\text{FIB}(n-2) & \quad \text{FIB}(n-3) \\
& \quad \text{SUM} \\
& \quad \text{FIB}(n-2) \\
& \quad \text{SUM} \\
\end{align*}
\]
Scheduling Parallel Programs

- **Problem**
  - Dependency between tasks
  - Size of tasks unknown
  - Number of tasks unknown

- **Goals**
  1. Be close to the optimal time.
  2. Know the influence of the machine architecture and the program characteristics.

Tell the programmer what parallel programs can be scheduled efficiently on which machine.
List scheduling: a first attempt

Use a greedy scheduler

- When tasks are available, no processor is idle.
List scheduling: a first attempt

Use a greedy scheduler

- When tasks are available, no processor is idle.

How to implement this scheduler?

- Store tasks in a global list
- Tasks generated by a running task are inserted in the list
- When a processor is idle, it retrieves a task from the list
List scheduling: a first attempt

Use a greedy scheduler

- When tasks are available, no processor is idle.

How to implement this scheduler?

- Store tasks in a global list
- Tasks generated by a running task are inserted in the list
- When a processor is idle, it retrieves a task from the list

Problem: contention on the list

- The list is accessed concurrently by several processors
- Protect the list by a lock or use a lock-free list
- Does not perform well in practice for small grain tasks
How to manage the tasks efficiently?
How to manage the tasks efficiently?

**Decentralize the list and use work stealing**
- Each processor has its own list
- Tasks generated are inserted in a local list
- When a processor is idle, it first checks its own list
- If empty, it tries to steal tasks in others’ lists

- When processor 3 is idle.
- It chooses another processor at random (for example, proc 2).
- It *steals* some tasks from 2.
How to manage the tasks efficiently?

Decentralize the list and use work stealing

- Each processor has its own list
- Tasks generated are inserted in a local list
- When a processor is idle, it first checks its own list
- If empty, it tries to steal tasks in others’ lists

- When processor 3 is idle.
- It chooses another processor at random (for example, proc 2).
- It steals some tasks from 2.
How to manage the tasks efficiently?

**Decentralize the list and use work stealing**

- Each processor has its own list
- Tasks generated are inserted in a local list
- When a processor is idle, it first checks its own list
- If empty, it tries to steal tasks in others’ lists

- When processor 3 is idle.
- It chooses another processor at random (for example, proc 2).
- It *steals* some tasks from 2.

- Reduce contention on lists
How to manage the tasks efficiently?

**Decentralize the list and use work stealing**

- Each processor has its own list
- Tasks generated are inserted in a local list
- When a processor is idle, it first checks its own list
- If empty, it tries to steal tasks in others’ lists

Reduce contention on lists

The scheduler is no longer greedy:
A processor can be idle while tasks are available in others’ lists.
Previous Work: Work Stealing

- Study of the makespan on identical processors
  [Blumofe Leiserson 99, Arora et al. 01]
  \[\mathbb{E}[C_{\text{max}}] \leq \frac{W}{m} + O(D)\]
  - DAG with only 1 source and out-degree at most 2
  - Big constant factor (32 compared to 3 in reality).

- Steady state results (stability and performance)
  [Mitzenmacher 98, Berenbrink et al. 03]
  - Very limited model

In this talk

Two problems and two methods:
- Total completion time using a potential function.
- Impact of different factors using mean field theory.
Outline of the talk

Overview

1. Total completion time analysis: the price of decentralization
   1. A adversary/potential method
   2. The price of decentralization
   3. Taking precedences into account.

2. Influence of some factors
   1. A generic model of work stealing in grid computing.
   2. Building a mean field model.
   3. Numerical study of this model

3. Conclusion
Model and notations

- Platform with $m$ synchronized and identical processors.
- Workload of $W$ independent unit tasks.

Work stealing algorithm

Time is discrete. At each time step:

- An active processor (non-empty list) executes one unit of work
- An idle processor:
  - randomly chooses another processor (victim)
  - If the victim’s list is non empty, the thief steals half of the tasks and resumes execution at the next time slot
  - Otherwise, the thief tries again at the next time slot

To model contention on lists:
if several thieves target the same victim a random succeed, others fail.
Analysis of the number of steals

Gantt chart with 25 processors and 2000 unit tasks
White: execution  Grey: steal

Property
At each time step, a processor is either working or stealing. Thus:

\[ m \cdot C_{\text{max}} = W + S \]

- **Problem**: Difficult to see any structure due to the random choices
A potential function approach

Notations:
- $w_i(t)$ – number of tasks of $i$ at time $t$
- $w(t) = \sum w_i(t)$ – total number of tasks at time $t$
- $r(t)$ – number of idle processors at time $t$

Definition

$$\Phi(t) = \sum_{1 \leq i \leq m} \left( w_i(t) - \frac{w(t)}{m} \right)^2$$

$\Phi$ represents how well the load is balanced between processors:
Potential Function $\Phi$: Properties

$$
\Phi(t) = \sum_{1 \leq i \leq m} \left( w_i(t) - \frac{w(t)}{m} \right)^2
$$

1. $\Phi = 0 \implies$ no more steals
Potential Function $\Phi$: Properties

$$\Phi(t) = \sum_{1 \leq i \leq m} \left( w_i(t) - \frac{w(t)}{m} \right)^2$$

1. $\Phi = 0 \implies$ no more steals

2. $\Phi$ decreases when executing tasks.
Potential Function $\Phi$: Properties

$$\Phi(t) = \sum_{1 \leq i \leq m} \left( w_i(t) - \frac{w(t)}{m} \right)^2$$

1. $\Phi = 0 \implies$ no more steals

2. $\Phi$ decreases when executing tasks.

3. If $j$ is being stolen: potential decreases by $\frac{w_j^2}{2}$
Expected decrease of $\Phi$

- If $i$ is stolen: potential decrease by $\frac{w_i(t)^2}{2}$.
- If there are $r(t)$ idle processors ($= r(t)$ steal requests) at time $t$: $i$ is stolen with probability $p(r(t)) = 1 - (1 - \frac{1}{m-1})^{r(t)}$.
- The expected decrease of the potential is $p(r(t)) \sum_i \frac{w_i(t)^2}{2}$.

Property

There exists $h(r)$ such that:

$$\mathbb{E}[\Phi(t + 1)] \leq h(r(t)) \cdot \Phi(t)$$

We want to compute the number of steals $S$:

$$S = \sum_{t=1}^{T} r(t) \quad \text{where} \quad T = \inf\{t \text{ such that } \Phi(t) = 0\}.$$
Expected decrease of $\Phi$

- If $i$ is stolen: potential decrease by $\frac{w_i(t)^2}{2}$.
- If there are $r(t)$ idle processors ($= r(t)$ steal requests) at time $t$: $i$ is stolen with probability $p(r(t)) = 1 - (1 - \frac{1}{m-1})^{r(t)}$.
- The expected decrease of the potential is $p(r(t)) \sum_i \frac{w_i(t)^2}{2}$.

**Property**

There exists $h(r)$ such that:

$$\mathbb{E}[\Phi(t + 1)] \leq h(r(t)) \cdot \Phi(t)$$

We want to compute the number of steals $S$:

$$S = \sum_{t=1}^{T} r(t) \quad \text{where} \quad T = \inf\{t \text{ such that } \Phi(t) = 0\}.$$  

Problem: we don't know $r(t)$
An adversary potential method

We define a Markov decision process:

An adversary chooses the worst sequence of \( r(t) \) to maximize \( \sum_{t=1}^{T} r(t) \)

subject to:
- \( \mathbb{E}[\Phi(t+1)] \leq h(r(t)) \cdot \Phi(t) \) and
- \( T = \inf\{t \text{ such that } \Phi(t) = 0\} \)

This leads to:

**Theorem**

\[
\mathbb{E}[S] \leq 1.83 \log_2 \Phi(0) + 3.63 \\
\leq 3.65 \log_2 W + 3.63.
\]

\[
\mathcal{P}(S \geq 1.83 (2 \log_2 W + \log_2 \epsilon) + 1) \leq \epsilon.
\]

Where 1.83 is an approximation of \( \frac{1}{1-\log_2(1+\frac{1}{\epsilon})} \) and 3.63 = 1.83/ln2 + 1.
**Bound on the total completion time**

Using that $C_{\text{max}} \cdot m = W + S$, we have:

**Theorem**

*The time for executing $W$ unit tasks by $m$ processor using work stealing is bounded by:*

$$\frac{W}{m} + 3.65 \log W + 3.63$$

In fact, using an other potential function $\Phi = \sum_i w_i(t)^{2.94}$, we show that:

$$\frac{W}{m} + 3.24 \log W + 3.33$$

**The price of decentralization**

A centralized scheduler cannot be better than $\frac{W}{m}$. We introduce the **price of decentralization**:

$$C_{\text{max}}^{\text{distributed}} - C_{\text{max}}^{\text{centralized}} \leq 3.24 \log W + 3.33.$$
# Price of decentralization for independent tasks

Time to schedule $n$ tasks of total processing time $W$:

<table>
<thead>
<tr>
<th></th>
<th>Centralized</th>
<th>Price of decentralization (WS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit Tasks ($W = n$)</strong></td>
<td>$\left\lceil \frac{W}{m} \right\rceil$</td>
<td>$3.24 \log_2 W + 3.4$</td>
</tr>
<tr>
<td>Initial repartition</td>
<td>$-$</td>
<td>$1.83 \log_2 \sum_{i=0}^{m} (w_i - \frac{W}{m})^2 + 3.63$</td>
</tr>
<tr>
<td>Cooperative</td>
<td>$-$</td>
<td>$2.88 \log_2 W + 3.07$</td>
</tr>
<tr>
<td><strong>Weighted Tasks</strong></td>
<td>$\frac{W}{m} + \frac{m - 1}{m} p_{\text{max}}$</td>
<td>$3.24 \log_2 n + 3.4$</td>
</tr>
</tbody>
</table>

Simulation indicates that the bounds are almost tight (real $\approx 2.37$ while our analysis shows 3.24 (gap: adversary)).
Unit tasks with precedence constraints

DAG Model
- DAG is discovered online
- At the beginning, all sources are placed in the lists (tasks in level 1)
- Tasks activated at step $t$ are placed in the lists and can be executed at step $t + 1$
- Level by level decomposition of the DAG
  Level $i$ contains $L_i$ tasks

Processor Algorithm
- If local list is non empty, execute one task
- If empty, steal half the tasks of the highest level on a random processor
Bound: DAG with max inter level dependencies

- A DAG with $W$ tasks and critical path $D$
- Number of tasks at level $i$: $L_1, \ldots, L_D$

**Property**

The total completion time is bounded by the one of a DAG with max inter level dependency:

\[ T \leq W \max_{1 \leq i \leq D} \left( \log_2 L_i + 3 \right) + 3 \cdot 4 D \]

Instead of $3 \cdot 24 \log_2 W + 3 \cdot 4 D$ for independent tasks.
**Bound: DAG with max inter level dependencies**

- A DAG with $W$ tasks and critical path $D$  
- Number of tasks at level $i$: $L_1, \ldots, L_D$

**Property**

The total completion time is bounded by the one of a DAG with max inter level dependency:

The total completion time is bounded by:

$$\mathbb{E}[C_{\text{max}}] \leq \frac{W}{m} + 3.24 \sum_{1 \leq i \leq D} \log_2 L_i + 3.4D$$

Instead of $3.24 \log_2 W + 3.4$ for independent tasks.
Overview

1. Total completion time analysis: the price of decentralization
   1. A adversary/potential method
   2. The price of decentralization
   3. Taking precedences into account.

2. Influence of some factors
   1. A generic model of work stealing in grid computing.
   2. Building a mean field model.
   3. Numerical study of this model

3. Conclusion
Goal of this part

Build a high level model of WS in order to conduct a precise study of the influence of some parameters.

- High Level Model

- Independent of a target architecture
- High Expressive Power
  - Variants of WS
  - Many parameters
- Important:
  - Keep it simple
  - Easy to study
**Goal of this part**

Build a **high level model** of WS in order to conduct a precise study of the influence of some parameters.

- Independent of a target architecture
- High Expressive Power
  - Variants of WS
  - Many parameters
- Important:
  - Keep it simple
  - Easy to study
- **Impact of stealing latency.**
- Extreme values
- **Strategy of stealing** (which processor to steal from)
- Impact of **stealing fraction**
Work stealing model

- $C$ clusters.

For each cluster:
- $N \cdot x_c$ processors.
- Arrival rate $\lambda_c$
- Speed $\mu_c$
- Steals from some cluster $c'$
  - $c'$ is chosen with proba. $p_{cc'}$
  - stealing rate $\gamma_{cc'}$ (does no depend on the number of jobs stolen).
Mean field principle

Mean field principle: The behavior of a complex system made of \( N \) objects becomes simpler when \( N \) goes to infinity when it is symmetric.
Mean field principle

Mean field principle: The behavior of a complex system made of \( N \) objects becomes simpler when \( N \) goes to infinity when it is symmetric.

- \( N \) processors in state \( P_1 \ldots P_N \).
- \((P_1 \ldots P_N)\) is a Markov chain.

Dynamics of the system is invariant by permutation of the objects:
\[
(P_1, \ldots, P_N) =_{\text{db}} (P_{\sigma(1)}, \ldots, P_{\sigma(N)}).
\]

Proportion of objects in state \( s \) is:
\[
X_s^N = \frac{1}{N} \sum_{n=1}^{N} 1_{P_n = s}
\]

Proposition
\[
X^N = (X_1^N, \ldots, X_S^N) \text{ has the Markov property.}
\]
### Mean field principle

**Mean field principle**: The behavior of a complex system made of \( N \) objects becomes simpler when \( N \) goes to infinity when it is symmetric.

- \( N \) processors in state \( P_1 \ldots P_N \).
- \( (P_1 \ldots P_N) \) is a Markov chain.

Dynamics of the system is invariant by permutation of the objects:

\[
(P_1, \ldots, P_N) =_{db} (P_{\sigma(1)}, \ldots, P_{\sigma(N)}).
\]

Proportion of objects in state \( s \) is:

\[
X_s^N = \frac{1}{N} \sum_{n=1}^{N} 1_{P_n = s}
\]

**Proposition**

\( X^N = (X_1^N, \ldots, X_S^N) \) has the Markov property.

**Theorem (Kurtz 79)**

- \( X^N(t) \rightarrow x(t) \) uniformly on \([0, T]\) where \( x \) is solution of a deterministic ODE \( \dot{x} = f(x) \).
- If the mean field has a unique attractor \( x^* \), the stationary measure \( \pi^N \) of \( X^N \) converges to the Dirac measure \( \delta_{x^*} \).
Mean field limit for work stealing

$X_{cj}^N$ is the proportion of proc. in cluster $c$ having $j$ jobs.

$X_{c0c'}^N$ is the proportion of proc. in cluster $c$ with 0 jobs stealing from $c'$.

$$
\begin{align*}
\left( X_{cj}^N(t), X_{c0c'}^N(t) \right) &\rightarrow (x_{cj}(t), x_{c0c'}(t)) \\
\text{for } &
\end{align*}
$$

\begin{align*}
\dot{x}_{c0c'} &= \mu_c x_{c1} p_{cc'} - (\lambda_c + \gamma_{cc'}) x_{c0c'} + \sum_{c''} \gamma_{cc''} x_{c0c''} \frac{x_{c''0} + x_{c''1}}{x_{c''}} p_{cc'} \\
\dot{x}_{c1} &= \mu_c x_{c2} - (\mu_c + \lambda_c) x_{c1} + \sum_{c'} \lambda_c x_{c0c'} + \sum_{c'} \gamma_{c'c'c'0c} x_{c0c} x_{c2} / x_c \\
&\quad + \sum_{c'} \gamma_{cc'} x_{c0c'} (x_{c'2} + x_{c'3}) / x_{c'} \\
\dot{x}_{cj} &= -(\mu_c + \lambda_c 1_{j<K}) x_{c,j} + \mu_c x_{c,j+1} + \lambda_c x_{c,j-1} \\
&\quad + \sum_{c'} \gamma_{c'c'c'0c} (x_{c,2j} + x_{c,2j-1}) / x_c \\
&\quad + \sum_{c'} \gamma_{cc'} x_{c0c'} (x_{c',2j} + x_{c',2j+1}) / x_{c'} \\
&\quad - \sum_{c'} \gamma_{c'c'c'0c} x_{cj} / x_c,
\end{align*}

- $N \cdot x_c$ processors.
- Arrival rate $\lambda_c$
- Speed $\mu_c$
- Steal from cluster $c'$ with proba. $p_{cc'}$
- Stealing rate $\gamma_{cc'}$ (does not depend on the number of jobs stolen).
Mean field limit: order of convergence is $O(1/\sqrt{N})$
Average sojourn time as a function of the rate of stealing $\gamma$ for various values of $\lambda$ (.3, .7 and .9).

- Sojourn time decreases from $1/(1 - \lambda)$ to $1/\mu = 1$.
- When $\gamma$ is small, it decreases dramatically.

More results in the paper [G., Gaujal, 10]
Fast Simulation: beyond the average values.

What about distributions of performance metrics?

- We want to study $P_i^N(t)$, the state of one particular processor.
- But: $P_i^N$ is a complicated random process (depends on $X^N$).
Fast Simulation: beyond the average values.

What about distributions of performance metrics?
- We want to study $P_i^N(t)$, the state of one particular processor.
- But: $P_i^N$ is a complicated random process (depends on $X^N$).

**Theorem (Fast Simulation)**

As $N$ grows, $P_i^N(t)$ goes to a non-homogeneous Markov chain of Kernel $K(x)$ where:
- $x(t)$ satisfies an ordinary differential equation $\dot{x} = f(x)$.

**Fast Simulation Algorithm for Steady State**
- If there exists a unique attractor $x^*$, $x(t)$ is constant and $x(t) = x^*$.
- We can simulate $P_i^N$:
  - $P_i^N(0)$ is picked according to law $x^*$.
  - $P_i^N$ is a markov chain of kernel $K(x^*)$. 
Fast simulation: distribution of sojourn times

99th percentiles of the sojourn time and of an exponential variable of the same mean, as functions of $\gamma$, for $\lambda = .7$.

- the tail of distribution of sojourn time is lighter than an exponential.
Work stealing in heterogeneous clusters

We consider an heterogeneous system.

- Study strategy of stealing in multiple scenarios:
  - Several Homogeneous clusters interconnected with a slower network.
  - Several Heterogeneous clusters
  - Master-Worker paradigm and probabilistic stealing.

In the next slides, we show that:

- Optimal stealing strategies depend on the load rather that on the latency.
- Master/worker is a good setting.
Heterogeneous clusters

Figure: Average sojourn time as a function of $p_{00}$ for the two heterogeneous model. The first cluster is lightly loaded ($\lambda_0 = .5$). The load of the second cluster is $\lambda_1$ (varying from .8 to 1.1).
Overview

1. Total completion time analysis: the price of decentralization
   1. A adversary/potential method
   2. The price of decentralization
   3. Taking precedences into account.

2. Influence of some factors
   1. A generic model of work stealing in grid computing.
   2. Building a mean field model.
   3. Numerical study of this model

3. Conclusion
**Conclusion and perspectives**

Two different methodology

- Usage of a potential function (Also apply to coupon collector problem).

\[ C_{\text{max}} = \frac{W}{m} + 3.24 \log_2 W + 3.4 \]

\[ \Downarrow \]

- Centralized time
- Price of decentralization

- Mean field model:
  - The time of simulation for all curves is less than 5min.
  - One can study distribution via fast simulation algorithm.

**Perspectives:**

- Study steady state behavior with arrival of bag of tasks.
- Energy-efficient work-stealing.
- Identify other DAG characteristics.
Conclusion and perspectives

Two different methodology

- Usage of a potential function (Also apply to coupon collector problem).

\[ C_{\text{max}} = \frac{W}{m} + 3.24 \log_2 W + 3.4 \]

- Mean field model:
  - The time of simulation for all curves is less than 5min.
  - One can study distribution via fast simulation algorithm.

Perspectives:

- Study steady state behavior with arrival of bag of tasks.
- Energy-efficient work-stealing.
- Identify other DAG characteristics.

Thank you for your attention.