DECENTRALIZED LIST SCHEDULING

MARC TCHIBOUKDJIAN, NICOLAS GAST, AND DENIS TRYSTRAM

ABSTRACT. Classical list scheduling is a very popular and efficient technique for scheduling jobs in parallel and distributed platforms. It is inherently centralized. However, with the increasing number of processors, the cost for managing a single centralized list becomes too prohibitive. A suitable approach to reduce the contention is to distribute the list among the computational units: each processor has only a local view of the work to execute. Thus, the scheduler is no longer greedy and standard performance guarantees are lost.

The objective of this work is to study the extra cost that must be paid when the list is distributed among the computational units. We first present a general methodology for computing the expected makespan based on the analysis of an adequate potential function which represents the load unbalance between the local lists. We obtain an equation on the evolution of the potential by computing its expected decrease in one step of the schedule. Our main theorem shows how to solve such equations to bound the makespan. Then, we apply this method to several scheduling problems, namely, for unit independent tasks, for weighted independent tasks and for tasks with precedence constraints. More precisely, we prove that the time for scheduling a global workload W composed of independent unit tasks on m processors is equal to W/m plus an additional term proportional to $\log_2 W$. We provide a lower bound which shows that this is optimal up to a constant. This result is extended to the case of weighted independent tasks. In the last setting, precedence task graphs, our analysis leads to an improvement on the bound of Arora et al (2001). We finally provide some experiments using a simulator. The distribution of the makespan is shown to fit existing probability laws. Moreover, the simulations give a better insight on the additive term whose value is shown to be around $3\log_2 W$ confirming the tightness of our analysis.

1. INTRODUCTION

1.1. **Context and motivations.** Scheduling is a crucial issue while designing efficient parallel algorithms on new multi-core platforms. The problem corresponds to distribute the tasks of an application (that we will called load) among available computational units and determine at what time they will be executed. The most common objective is to minimize the completion time of the latest task to be executed (called the *makespan* and denoted by $C_{\max}$). It is a hard challenging problem which received a lot of attention during the last decade (Leung, 2004). Two new books have been published recently on the topic (Drozdowski, 2009; Robert and Vivien, 2009), which confirm how active is the area.

List scheduling is one of the most popular technique for scheduling the tasks of a parallel program. This algorithm has been introduced by Graham (1969) and was used with profit in many further works (for instance the earliest task first heuristic which extends the analysis for communication delays in Hwang et al (1989), for uniform machines in Chekuri and Bender (2001), or for parallel rigid jobs in Schwiegelshohn et al (2008)). Its principle is to build a list of ready tasks and schedule them as soon as there exist available resources. List scheduling algorithms are low-cost (greedy) whose performances are not too far from

1

optimal solutions. Most proposed list algorithms differ in the way of considering the priority of the tasks for building the list, but they always consider a centralized management of the list. However, today the parallel and distributed platforms involve more and more processors. Thus, the time needed for managing such a centralized data structure can not be ignored anymore. Practically, implementing such schedulers induces synchronization overheads when several processors access the list concurrently. Such overheads involve low-level synchronization mechanisms.

1.2. **Related works.** Most related works dealing with scheduling consider centralized list algorithms. However, at execution time, the cost for managing the list is neglected. To our knowledge, the only approach that takes into account this extra management cost is *work stealing* (Blumofe and Leiserson, 1999) (denoted by WS in short).

Contrary to classical centralized scheduling techniques, WS is by nature a distributed algorithm. Each processor manages its own list of tasks. When a processor becomes idle, it randomly chooses another processor and *steals* some work. To model contention overheads, processors that request work on the same remote list are in competition and only one can succeed. WS has been implemented in many languages and parallel libraries including Cilk (Frigo et al, 1998), TBB (Robison et al, 2008) and KAAPI (Gautier et al, 2007). It has been analyzed in a seminal paper of Blumofe and Leiserson (1999) where they show that the expected makespan of series-parallel precedence graph with W unit tasks on m processors is bounded by $\mathbb{E}\left[C_{\max}\right] \leq W/m + \mathrm{O}(D)$ where D is the critical path of the graph (its depth). This analysis has been improved in Arora et al (2001) using a proof based on a potential function. The case of varying processor speeds has been analyzed in Bender and Rabin (2002). However, in all these previous analyses, the precedence graph is constrained to have only one source and out-degree at most 2 which does not easily model the basic case of independent tasks. Simulating independent tasks with a binary tree of precedences gives a bound of $W/m + \mathrm{O}(\log W)$ as a complete binary tree of W nodes has a depth of $D \leq \log_2 W$. However, with this approach, the structure of the binary tree dictates which tasks are stolen. Our approach achieves a bound of the same order with a better constant and processors are free to choose which tasks to steal. Notice that there exist other ways to analyze work stealing where the work generation is probabilist and that targets steady state results (Berenbrink et al, 2003; Mitzenmacher, 1998; Gast and Gaujal, 2010).

Another related approach which deals with distributed load balancing is *balls into bins* games (Azar et al, 1999; Berenbrink et al, 2008). The principle is to study the maximum load when n balls are randomly thrown into m bins. This is a simple distributed algorithm which is different from the scheduling problems we are interested in. First, it seems hard to extend this kind of analysis for tasks with precendence constraints. Second, as the load balancing is done in one phase at the beginning, the cost of computing the schedule is not considered. Adler et al (1995) study parallel allocations but still do not take into account contention on the bins. Our approach, like in WS, considers contention on the lists.

Some works have been proposed for the analysis of algorithms in data structures and combinatorial optimization (including variants of scheduling) using potential functions. Our analysis is also based on a potential function representing the load unbalance between the local queues. This technique has been successfully used for analyzing convergence to Nash equilibria in game theory (Berenbrink et al, 2007), load diffusion on graphs (Berenbrink et al, 2009) and WS (Arora et al, 2001).

1.3. **Contributions.** List scheduling is centralized in nature. The purpose of this work is to study the effects of decentralization on list scheduling. The main result is a new framework for analyzing distributed list scheduling algorithms (DLS). Based on the analysis of the load balancing between two processors during a work request, it is possible to deduce the total expected number of work requests and then, to derive a bound on the expected makespan.

This methodology is generic and it is applied in this paper on several relevant variants of the scheduling problem.

- We first show that the expected makespan of DLS applied on W unit independent tasks is equal to the absolute lower bound W/m plus an additive term in $3.65 \log_2 W$. We propose a lower bound which shows that the analysis is tight up to a constant factor. This analysis is refined and applied to several variants of the problem. In particular, a slight change on the potential function improves the multiplicative factor from 3.65 to 3.24. Then, we study the possibility of processors to cooperate while requesting some tasks in the same list. Finally, we study the initial repartition of the tasks and show that a balanced initial allocation induces less work requests.
- Second, the previous analysis is extended to the weighted case of any unknown processing times. The analysis achieves the same bound as before with an extra term involving $p_{\max}$ (the maximal value of the processing times).
- Third, we provide a new analysis for the WS algorithm of Arora et al (2001) for scheduling DAGs that improves the bound on the number of work requests from $32mD$ to $5.5mD$.
- Fourth, we developed a complete experimental campaign that gives statistical evidence that the makespan of DLS follows known probability distributions depending on the considered variant. Moreover, the experiments show that the theoretical analysis for independent tasks is almost tight: the overhead to W/m is less than 37% away of the exact value.

1.4. **Content.** We start by introducing the model and we recall the analysis for classical list scheduling in Section 2. Then, we present the principle of the analysis in Section 3 and we apply this analysis on unit independent tasks in Section 4. Section 5 discusses variations on the unit tasks model: improvements on the potential function and cooperation among thieves. We extend the analysis for weighted independent tasks in Section 6 and for tasks with dependencies in Section 7. Finally, we present and analyze simulation experiments in Section 8.

2. MODEL AND NOTATIONS

2.1. **Platform and workload characteristics.** We consider a parallel platform composed of m identical processors and a workload of n tasks with processing times p_j. The total work of the computation is denoted by $W = \sum_{j=1}^{n} p_j$. The tasks can be independent or constrained by a directed acyclic graph (DAG) of precedences. In this case, we denote by D the critical path of the DAG (its depth). We consider an online model where the processing times and precedences are discovered during the computation. More precisely, we learn the processing time of a task when its execution is terminated and we discover new tasks in the DAG only when all their precedences have been satisfied. The problem is to study the maximum completion time (*makespan* denoted by $C_{\max}$) taking into account the scheduling cost.
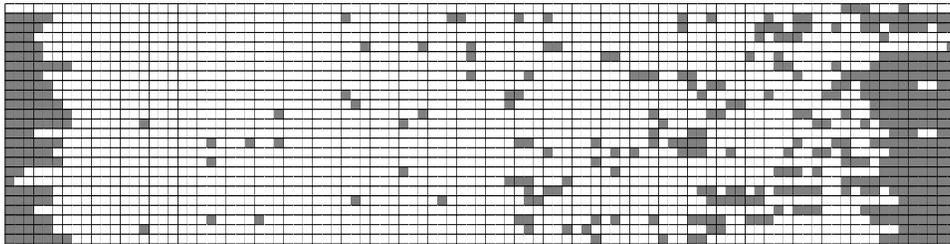
FIGURE 1. A typical execution of $W = 2000$ unit independent tasks on $m = 25$ processors using distributed list scheduling. Grey area represents idle times due to steal requests.

2.2. **Centralized list scheduling.** Let us recall briefly the principle of list scheduling as it was introduced by Graham (1969). The analysis states that the makespan of any list algorithm is not greater than twice the optimal makespan. One way of proving this bound is to use a geometric argument on the Gantt chart: $m \cdot C_{\max} = W + S_{\mathrm{idle}}$ where the last term is the surface of idle periods (represented in grey in figure 1).

Depending on the scheduling problem (with or without precedence constraints, unit tasks or not), there are several ways to compute S_{idle}. With precedence constraints, $S_{\mathrm{idle}} \leq (m-1) \cdot D$. For independent tasks, the results can be written as $S_{\mathrm{idle}} \leq (m-1) \cdot p_{\max}$ where $p_{\max}$ is the maximum of the processing times. For unit independent tasks, it is straightforward to obtain an optimal algorithm where the load is evenly balanced. Thus $S_{\mathrm{idle}} \leq m - 1$, *i.e.* at most one slot of the schedule contains idle times.

2.3. **Decentralized list scheduling.** When the list of ready tasks is distributed among the processors, the analysis is more complex even in the elementary case of unit independent tasks. In this case, the extra S_{idle} term is induced by the distributed nature of the problem. Processors can be idle even when ready tasks are available. Fig. 1 is an example of a schedule obtained using distributed list scheduling which shows the complicated repartition of the idle times S_{idle}.

2.4. **Model of the distributed list.** We now describe precisely the behavior of the distributed list. Each processor i maintains its own local queue Q_i of tasks ready to execute. At the beginning of the execution, ready tasks can be arbitrarily spread among the queues. While Q_i is not empty, processor i picks a task and executes it. When this task has been executed, it is removed from the queue and another one starts being processed. When Q_i is empty, processor i sends a *steal request* to another processor k chosen uniformly at random. If Q_k is empty or contains only one task (currently executed by processor k), then the request fails and processor i will send a new request at the next time step. If Q_k contains more than one task, then i is given half of the tasks and it will restart a normal execution at the next step. To model the contention on the queues, no more than one steal request per processor can succeed in the same time slot. If several requests target the same processor, a random one succeeds and all the others fail. This assumption will be relaxed in Section 5.2. A steal request is said *successful* if the target queue contains more than one task and the request is not aborted due to contention. In all the other cases, the steal request is said *unsuccessful*.

This is a high level model of a distributed list but it accurately models the case of independent tasks and the WS algorithm of Arora et al (2001). We justify here some choices of this model. There is no explicit communication cost since WS algorithms most

often target shared memory platforms. In addition, a steal request is done in constant time independently of the amount of tasks transfered. This assumption is not restrictive as the description of a large number of tasks can be very short. In the case of independent tasks, a whole subpart of an array of tasks can be represented in a compact way by the range of the corresponding indices, each cell containing the effective description of a task (a STL transform in Traoré et al (2008)). For more general cases with precedence constraints, it is usually enough to transfer a task which represents a part of the DAG. More details on the DAG model are provided in Section 7. Finally, there is no contention between a processor executing a task from its own queue and a processor stealing in the same queue. Indeed, one can use queue data structures allowing these two operations to happen concurrently (Frigo et al, 1998).

2.5. **Properties of the work.** At time t, let $w_i(t)$ represent the amount of work in queue Q_i (*cf.* Fig. 2). $w_i(t)$ may be defined as the sum of processing times of all tasks in Q_i as in Section 4 but can differ as in Sections 6 and 7. In all cases, the definition of $w_i(t)$ satisfies the following properties.

  (1) When $w_i(t) > 0$, processor i is active and executes some work: $w_i(t+1) \leq w_i(t)$.
  (2) When $w_i(t) = 0$, processor i is idle and send a steal request to a random processor k. If the steal request is successful, a certain amount of work is transfered from processor k to processor i and we have $\max\{w_i(t+1), w_k(t+1)\} < w_k(t)$.
  (3) The execution terminates when there is no more work in the system, *i.e.* $\forall i, w_i(t) = 0$.

We also denote the total amount of work on all processors by $w(t) = \sum_{i=1}^{m} w_i(t)$ and the number of processors sending steal requests by $r_t \in [0, m-1]$. Notice that when $r_t = m$, all queues are empty and thus the execution is complete.
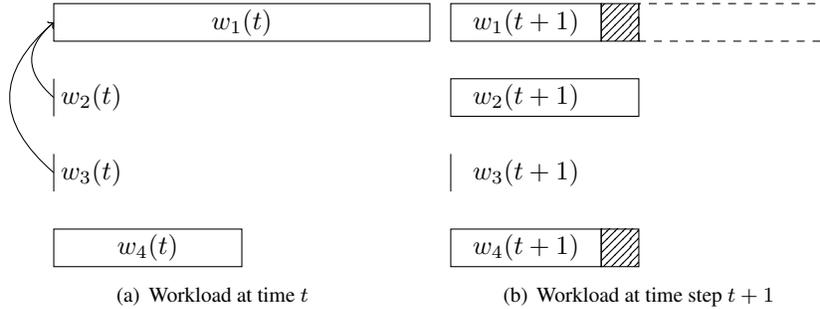

(a) Workload at time t (b) Workload at time step $t+1$

FIGURE 2. Evolution of the workload of the different processors during a time step. At time t, processors 2 and 3 are idle and they both choose processor 1 to steal from. At time $t + 1$, only processor 2 succeed in stealing some of the work of processor 1. The work is split between the two processors. Processors 1 and 4 both execute some work during this time step (represented by a shaded zone).

3. PRINCIPLE OF THE ANALYSIS AND MAIN THEOREM

This section presents the principle of the analysis. The main result is Theorem 1 that gives bounds on the expectation of the steal requests done by the schedule as well as the

probability that the number of work requests exceeds this bound. As a processor is either executing or requesting work, the number of work requests plus the total amount of tasks to be executed is equal to $m \cdot C_{\max}$, where $C_{\max}$ is the total completion time. The makespan can be derived from the total number of work requests:

$$(1) \qquad\qquad C_{\max} = \frac{W}{m} + \frac{R}{m}.$$

The main idea of our analysis is to study the decrease of a potential Φ_t. The potential Φ_t depends on the load on all processors at time t, $\mathbf{w}(t)$. The precise definition of Φ_t varies depending on the scenario (see Sections 4 to 7). For example, the potential function used in Section 4 is $\Phi_t = \sum_{i=1}^{m}(w_i(t) - w(t)/m)^2$. For each scenario, we will prove that the diminution of the potential during one time step depends on the number of steal requests, r_t. More precisely, we will show that there exists a function $h : \{0 \ldots m\} \to [0; 1]$ such that the average value of the potential at time $t + 1$ is less than $\Phi_t/h(r_t)$.

Using the expected diminution of the potential, we derive a bound on the number of steal requests until Φ_t becomes less than one, $R = \sum_{s=0}^{\tau-1} r_s$, where τ denotes the first time that Φ_t is less than 1. If all r_t were equal to r and the potential decrease was deterministic, the number of time steps before $\Phi_t \leq 1$ would be $\left\lceil \log \Phi_0 / \log h(r) \right\rceil$ and the number of steal requests would be $r/\log h(r) \log \Phi_0$. As r can vary between 1 and m, the worst case for this bound is $m\lambda \cdot \log \Phi_0$, where $m\lambda = \max_{1 \leq r \leq m} r/\log(h(r))$.

The next theorem shows that number of steal requests is indeed bounded by $m\lambda \log \Phi_0$ plus an additive term due to the stochastic nature of Φ_t. The fact that λ corresponds to the worst choice of r_t at each time step makes the bound looser than the real constant. However, we show in Section 8 that the gap between the obtained bound and the values obtained by simulation is small. Moreover, the computation of the constant λ is simple and makes this analysis applicable in several scenarios, such as the ones presented in Sections 4 to 7.

In the following theorem and its proof, we use the following notations. $\mathscr{F}_t$ denotes the knowledge of the system up to time t (namely, the filtration associated to the process $\mathbf{w}(t)$). For a random variable X, the conditional expectation of A knowing $\mathscr{F}_t$ is denoted $\mathbb{E}[X \mid \mathscr{F}_t]$. Finally, the notation $\mathbf{1}_A$ denotes the random variable equal to 1 if the event A is true and 0 otherwise. In particular, this means that the probability of an event A is $\mathbb{P}\{A\} = \mathbb{E}[\mathbf{1}_A]$.

Theorem 1. *Assume that there exists a function $h : \{0 \ldots m\} \to [0, 1]$ such that the potential satisfies:*

$$\mathbb{E}[\Phi_{t+1} \mid \mathscr{F}_t] \leq h(r_t) \cdot \Phi_t.$$

Let Φ_0 denotes the potential at time 0 and λ be defined as:

$$\lambda \stackrel{\text{def}}{=} \max_{1 \leq r \leq m} \frac{r}{-m \log_2(h(r))}$$

Let τ be the first time that Φ_t is less than 1, $\tau \stackrel{\text{def}}{=} \min\{t : \Phi_t < 1\}$. The number of steal requests until τ, $R = \sum_{s=0}^{\tau-1} r_s$, satisfies:

  (i) $\mathbb{P}\{R \geq m \cdot \lambda \cdot \log_2 \Phi(0) + m + u\} \leq 2^{-u/(m \cdot \lambda)}$

  (ii) $\mathbb{E}[R] \leq m \cdot \lambda \cdot \log_2 \Phi(0) + m(1 + \dfrac{\lambda}{\ln 2})$.

Proof. For two time steps $t \leq T$, we call R_t^T the number of steal requests between t and T:

$$R_t^T \overset{\text{def}}{=} \sum_{s=t}^{\min\{\tau,T\}-1} r_s.$$

The number of steal requests until $\Phi_t < 1$ is $R = \sum_{s=0}^{\tau-1} r_s = \lim_{T \to \infty} R_0^T$.

We show by a backward induction on t that for all $t \leq T$:

(2) $\qquad$ if $\Phi_t \geq 1$, then $\forall u \in \mathbb{R} : \mathbb{E}\left[\mathbf{1}_{R_t^T \geq m \cdot \lambda \cdot \log_2 \Phi_t + m + u} \mid \mathscr{F}_t\right] \leq 2^{-u/(m \cdot \lambda)}$.

For $t{=}T$, $R_T^T = 0$ and $\mathbb{E}\left[\mathbf{1}_{R_T^T \geq m \cdot \lambda \cdot \log_2 \Phi_t + m + u} \mid \mathscr{F}_t\right] = 0$. Thus, (2) is true for $t{=}T$.

Assume that (2) holds for some $t{+}1 \leq T$ and suppose that $\Phi_t \geq 1$. Let $u > 0$ (if $u \leq 0$ ...). Since $R_t^T = r_t + R_{t+1}^T$, the probability $\mathbb{P}\left\{R_t^T \geq m \cdot \lambda \cdot \log_2 \Phi_t + m + u \mid \mathscr{F}_t\right\}$ is equal to

(3) $\quad \mathbb{E}\left[\mathbf{1}_{R_t^T \geq m\lambda \log_2 \Phi_t + m + u} \mid \mathscr{F}_t\right] = \mathbb{E}\left[\mathbf{1}_{r_t + R_{t+1}^T \geq m\lambda \log_2 \Phi_t + m + u} \mid \mathscr{F}_t\right]$

(4) $\qquad\qquad\qquad\qquad = \mathbb{E}\left[\mathbf{1}_{r_t + R_{t+1}^T \geq m\lambda \log_2 \Phi_t + m + u}\mathbf{1}_{\Phi_{t+1} \geq 1} \mid \mathscr{F}_t\right]$

(5) $\qquad\qquad\qquad\qquad\quad + \mathbb{E}\left[\mathbf{1}_{r_t + R_{t+1}^T \geq m\lambda \log_2 \Phi_t + m + u}\mathbf{1}_{\Phi_{t+1} < 1} \mid \mathscr{F}_t\right]$

If $\Phi_{t+1} < 1$, then $R_{t+1}^T = 0$. Since $m \geq r_t$ and $\Phi_t \geq 1$, $m\lambda \log_2 \Phi_t + m + u - r_t \geq 0$. This shows that the term of Equation (5) is equal to zero. (4) is the probability that R_{t+1}^T is greater than

$$m\lambda \log_2 \Phi_t + m + u - r_t = m\lambda \log_2 \Phi_{t+1} + m + (u - r_t - m\lambda \log(\Phi_{t+1}/\Phi_t)$$

Therefore, using the induction hypothesis, (4) is equal to

$$\mathbb{E}\left[\mathbf{1}_{R_{t+1}^T \geq m\lambda \log_2 \Phi_t + m + u - r_t}\mathbf{1}_{\Phi_{t+1} > 1} \mid \mathscr{F}_t\right] = \mathbb{E}\left[2^{-\frac{u - r_t - m\lambda \log(\Phi_{t+1}/\Phi_t)}{m\lambda}}\mathbf{1}_{\Phi_{t+1} > 1} \mid \mathscr{F}_t\right]$$

$$= 2^{-\frac{u - r_t}{m\lambda}}\mathbb{E}\left[\frac{\Phi_{t+1}}{\Phi_t}\mathbf{1}_{\Phi_{t+1} > 1} \mid \mathscr{F}_t\right]$$

$$= 2^{-\frac{u - r_t}{m\lambda}}h(r_t)$$

$$= 2^{-\frac{u}{m\lambda}}2^{r_t/\lambda + \log_2(h(r_t))},$$

where at the first line we used both the fact that for a random variable X, $\mathbb{E}\left[X \mid \mathscr{F}_t\right] = \mathbb{E}\left[\mathbb{E}\left[X \mid \mathscr{F}_{t+1}\right] \mid \mathscr{F}_t\right]$ and the induction hypothesis.

If $r_t = 0$, $2^{r_t/\lambda + \log_2(h(r_t))} = h(r_t) \leq 1$. Otherwise, by definition of $\lambda = \max_{1 \leq r \leq m} r/- \log(h(r))$, $r_t/\lambda + \log_2(h(r_t)) \leq 0$ and $2^{r_t/\lambda + \log_2(h(r_t))} \leq 1$. This shows that (2) holds for t. Therefore, by induction on t, this shows that (2) holds for $t = 0$: for all $u \geq 0$:

$$\mathbb{E}\left[\mathbf{1}_{R_0^T \geq m \cdot \lambda \cdot \log_2 \Phi_t + m + u} \mid \mathscr{F}_0\right] \leq 2^{-u/(m \cdot \lambda)}$$

As $r_t \geq 0$, the sequence $(R_0^T)_T$ is increasing and converges to R. Therefore, the sequence $\mathbf{1}_{R_0^T \geq m \cdot \lambda \cdot \log_2 \Phi_0 + m + u}$ is increasing in T and converges to $\mathbf{1}_{R \geq m \cdot \lambda \cdot \log_2 \Phi_0 + m + u}$. Thus, by Lebesgue's monotone convergence theorem, this shows that

$$\mathbb{P}\left\{R \geq m \cdot \lambda \cdot \log_2 \Phi_0 + m + u\right\} = \lim_{T \to \infty} \mathbb{E}\left[\mathbf{1}_{R_0^T \geq m \cdot \lambda \cdot \log_2 \Phi_0 + m + u}\right] \leq 2^{-\frac{u}{m\lambda}}.$$

The second part of the theorem *(ii)* is a direct consequence of *(i)*. Indeed,

$$
\begin{aligned}
\mathbb{E}\left[R\right] &= \int_0^\infty \mathbb{P}\left\{R \geq u\right\} du \\
&\leq m \cdot \lambda \cdot \log_2 \Phi_0 + m + \int_0^\infty \mathbb{P}\left\{R \geq m \cdot \lambda \cdot \log_2 \Phi_0 + m + u\right\} du \\
&\leq m \cdot \lambda \cdot \log_2 \Phi_0 + m + \int_0^\infty 2^{-\frac{u}{m\lambda}} du \\
&\leq m \cdot \lambda \cdot \log_2 \Phi_0 + m(1 + \frac{\lambda}{\ln 2}).
\end{aligned}
$$

$\square$ $\square$

4. UNIT INDEPENDENT TASKS

We apply the analysis presented in the previous section for the case of independent unit tasks. In this case, each processor i maintains a local queue Q_i of tasks to execute. At every time slot, if the local queue Q_i is not empty, processor i picks a task and executes it. When Q_i is empty, processor i sends a steal request to a random processor j. If Q_j is empty or contains only one task (currently executed by processor j), then the request fails and processor i will have to send a new request at the next slot. If Q_j contains more than one task, then i is given half of the tasks (after that the task executed at time t by processor j has been removed from Q_j). The amount of work on processor i at time t, $w_i(t)$, is the number of tasks in $Q_i(t)$. At the beginning of the execution, $w(0) = W$ and tasks can be arbitrarily spread among the queues.

4.1. **Potential function and expected decrease.** Applying the method presented in Section 3, the first step of the analysis is to define the potential function and compute the potential decrease when a steal occurs. For this example, $\Phi(t)$ is defined by:

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

This potential represents the load unbalance in the system. If all queues have the same load $w_i(t) = w(t)/m$, then $\Phi(t) = 0$. $\Phi(t) \leq 1$ implies that there is at most one processor with at most one more task than the others. In that case, there will be no steal until there is just one processor with 1 task and all others idle. Moreover, the potential function is maximal when all the work is concentrated on a single queue. That is $\Phi(t) \leq w(t)^2 - w(t)^2/m \leq (1 - 1/m)w^2(t)$.

Three events contribute to a variation of potential: successful steals, tasks execution and decrease of $w^2(t)/m$.

(1) If the queue i has $w_i(t) \geq 1$ tasks and it receives one or more steal requests, it chooses a processor j among the thieves. At time $t + 1$, i has executed one task and the rest of the work is split between i and j. Therefore,

$$
w_i(t+1) = \left\lceil (w_i(t) - 1)/2 \right\rceil \quad \text{and} \quad w_j(t+1) = \left\lfloor (w_i(t) - 1)/2 \right\rfloor.
$$

Thus, we have:

$$
w_i(t+1)^2 + w_j(t+1)^2 = \left\lceil (w_i(t)-1)/2 \right\rceil^2 + \left\lfloor (w_i(t)-1)/2 \right\rfloor^2 \leq w_i(t)^2/2 - w_i(t) + 1.
$$

Therefore, this generates a difference of potential of

(6) $$\delta_i(t) \geq w_i(t)^2/2 + w_i(t) - 1.$$

(2) If i has $w_i(t) \geq 1$ tasks and receives zero steal requests, it potential goes from $w_i(t)^2$ to $(w_i(t) - 1)^2$, generating a potential decrease of $2w_i(t) - 1$.

(3) As there are $m - r_t$ active processors, $(\sum_{i=1}^{m} w_i(t))^2/m$ goes from $w(t)^2/m$ to $w(t+1)^2 = (w(t) - m + r)^2/m$, generating a potential increase of $2(m - r_t)w(t)/m - (m - r_t)^2/m$.

Recall that at time t, there are r_t processors that send steal requests. A processor i receives zero steal requests if the r_t thieves choose another processor. Each of these events is independent and happens with probability $(m - 2)/(m - 1)$. Therefore, the probability for the processor to receive one or more steal requests is $q(r_t)$ where

$$q(r_t) = 1 - \left(1 - \frac{1}{m-1}\right)^{r_t}.$$

If $\Phi_t = \Phi$ and $r_t = r$, by summing the expected decrease on each active processor δ_i, the expected potential decrease is greater than:

$$\sum_{i/w_i(t)>0} \left[\underbrace{q(r)\left(\frac{w_i(t)^2}{2} + w_i(t) - 1\right) + (1 - q(r))(2w_i(t) - 1)}_{\geq \delta_i} \right] - 2w(t)\frac{m-r}{m} + \frac{(m-r)^2}{m}$$

$$= \left[\sum_{i/w_i(t)>0} \frac{q(r)}{2}w_i(t)^2 \right] - q(r)w(t) + 2w(t) - (m - r) - 2w(t)\frac{m-r}{m} + \frac{(m-r)^2}{m}.$$

Using that $2w(t) - 2w(t)\frac{m-r}{m} = 2w(t)\frac{r}{m}$, that $-(m - r) + \frac{(m-r)^2}{m} = -(m - r)\frac{r}{m}$ and that $\sum w_i(t)^2 = \Phi + w(t)^2$, this equals:

$$\frac{q(r)}{2}\Phi + \frac{q(r)}{2}\frac{w(t)^2}{m} - q(r)w(t) + 2w(t)\frac{r}{m} - (m - r)\frac{r}{m}$$

$$= \frac{q(r)}{2}\Phi + \frac{q(r)}{2}\frac{w(t)^2}{m} - q(r)w(t) + \frac{r}{m}(2w(t) - m + r)$$

$$= \frac{q(r)}{2}\Phi + \frac{q(r)w(t)}{2}\left(\frac{w(t)}{m} - 2 + \frac{2r}{mq(r)}\right) + \frac{r}{m}(w(t) - m + r).$$

By concavity of $x \mapsto (1 - (1 - x)^r)$, $(1 - (1 - x)^r) \leq r \cdot x$. This shows that $q(r) = 1 - (1 - \frac{1}{m-1})^r \leq r/(m - 1)$. Thus, $r/q(r) \geq m - 1$. Moreover, as $m - r$ is the number of active processors, $w \geq m - r$ (each processor has at least one task). This shows that the expected decrease of potential is greater than:

$$\frac{q(r)}{2}\Phi + \frac{q(r)w(t)}{2}\left(\frac{w(t)}{m} - 2 + 2\frac{m-1}{m}\right) = \frac{q(r)}{2}\Phi + \frac{q(r)w(t)}{2m}(w(t) - 2).$$

If $w(t) \geq 2$, then the expected decrease of potential is greater than $q(r_t)\Phi_t/2$. If $w(t) < 2$, this means that $w(t) = 1$ and $w(t+1) = 0$ and therefore $\Phi_{t+1} = 0$. Thus, for all t:

$$(7) \qquad \mathbb{E}\left[\Phi_{t+1} \mid \mathscr{F}_t\right] \leq \left(1 - \frac{q(r_t)}{2}\right) \cdot \Phi_t.$$

4.2. **Bound on the makespan.** Using Theorem 1 of the previous section, we can solve equation (7) and conclude the analysis.

Theorem 2. *Let $C_{\max}$ be the makespan of $W = n$ unit independent tasks scheduled by DLS and $\Phi_0 \overset{\text{def}}{=} \sum_i (w_i - \frac{W}{m})^2$ the potential when the schedule starts. Then:*

(i) $\mathbb{E}\left[C_{\max}\right] \leq \dfrac{W}{m} + \dfrac{1}{1 - \log_2(1 + \frac{1}{e})} \cdot \left(\log_2 \Phi_0 + \dfrac{1}{\ln 2}\right) + 1$

(ii) $\mathbb{P}\left\{C_{\max} \geq \dfrac{W}{m} + \dfrac{1}{1 - \log_2(1 + \frac{1}{e})} \cdot \left(\log_2 \Phi_0 + \log_2 \dfrac{1}{\epsilon}\right) + 1\right\} \leq \epsilon$

In particular:

(iii) $\mathbb{E}\left[C_{\max}\right] \leq \dfrac{W}{m} + \dfrac{2}{1 - \log_2(1 + \frac{1}{e})} \cdot \left(\log_2 W + \dfrac{1}{2\ln 2}\right) + 1$

These bounds are optimal up to a constant factor in $\log_2 W$.

Proof. Equation (7) shows that $\mathbb{E}\left[\Phi_{t+1}|\mathscr{F}_t\right] \leq g(r_t)\Phi_t$ with $g(r) = 1 - q(r)/2$. Defining $\Phi'_t = \Phi_t/(1 - 1/(m-1))$, the potential function Φ'_t also satisfies (7). Therefore, Φ'_t satisfies the conditions of Theorem 1. This shows that the number of work requests R until $\Phi'_t < 1$ satisfies

$$\mathbb{E}\left[R\right] \leq m \cdot \lambda \log_2(\Phi_0) + m\left(1 + \dfrac{\lambda}{\ln 2}\right),$$

with $\lambda = \max_{1 \leq r \leq m-1} r/(-m \log_2 h(r))$. One can show that $r/(-m \log_2 h(r))$ is decreasing in r. Thus its minimum is attained for $r = 1$. This shows that $\lambda \leq 1/(1 - \log_2(1 + \frac{1}{e}))$.

The minimal non zero-value for Φ_t is when one processor has one task and the others zero. In that case, $\Phi_t = 1 - 1/(m-1)$. Therefore, when $\Phi'_t < 1$, this means that $\Phi_t = 0$ and the schedule is finished.

As pointed out in Equation (1), at each time step of the schedule, a processor is either computing one task or stealing work. Thus, the number of steal requests plus the number of tasks to be executed is equal to $m \cdot C_{\max}$, *i.e.* $m \cdot C_{\max} = W + R$. This shows that

$$\mathbb{E}\left[C_{\max}\right] \leq \dfrac{W}{m} + \dfrac{1}{1 - \log_2(1 + \frac{1}{e})} \cdot \left(\log_2 \Phi_0 + \dfrac{1}{\ln 2}\right) + 1.$$

This concludes the proof of *(i)*. The proof of the *(i)* applies *mutatis mutandis* to prove the bound in probability *(ii)* using Theorem 1 *(ii)*.

We now give a lower bound for this problem. Consider $W = 2^{k+1}$ tasks and $m = 2^k$ processors, all the tasks being on the same processor at the beginning. In the best case, all steal requests target processors with highest loads. In this case the makespan is $C_{\max} = k + 2$: the first $k = \log_2 m$ steps for each processor to get some work; one step where all processors are active; and one last step where only one processor is active. In that case, $C_{\max} \geq \dfrac{W}{m} + \log_2 W - 1$. □   □

This theorem shows that the factor before $\log_2 W$ is bounded by 1 and $2/(1 - \log_2(1 + 1/e)) < 3.65$. Simulations reported in Section 8 seem to indicate that the factor of $\log_2 W$ is slightly less than 3.65. This shows that the constants obtained by our analysis are sharp.

4.3. **Influence of the initial repartition of tasks.** In the worst case, all tasks are in the same queue at the beginning of the execution and $\Phi_0 = (W - W/m)^2 \leq W^2$. This leads to a bound on the number of work requests in $3.65m \log_2 W$ (see the item *(iii)* of Theorem 2). However, using bounds in terms of Φ_0, our analysis is able to capture the difference for the number of work requests if the initial repartition is more balanced. One can show that a more balanced initial repartition ($\Phi_0 \ll W^2$) leads to fewer steal requests on average.

Suppose for example that the initial repartition is a balls-and-bins assignment: each tasks is assigned to a processor at random. In this case, the initial number of tasks in queue

i, $w_i(0)$, follows a binomial distribution $\mathcal{B}(W, 1/m)$. The expected value of Φ_0 is:

$$\mathbb{E}\left[\Phi_0\right] = \sum_i \mathbb{E}\left[w_i^2\right] - \frac{W^2}{m} = \sum_i \left(\text{Var}\left[w_i\right] + \mathbb{E}\left[w_i\right]^2\right) - \frac{W^2}{m} = \left(1 - \frac{1}{m}\right) \cdot W$$

Since the number of work requests is proportional to $\log_2 \Phi_0$, this initial repartition of tasks reduces the number of steal requests by a factor of 2 on average. This leads to a better bound on the makespan in $W/m + 1.83 \log_2 W + 3.63$.

5. GOING FURTHER ON THE UNIT TASKS MODEL

In this section, we provide two different analysis of the model of unit tasks of the previous section. We first show how the use of a different potential function $\Phi_t = \sum_i w_i(t)^\nu$ (for some $\nu > 1$) leads to a better bound on the number of work requests. Then we show how cooperation among thieves leads to a reduction of the bound on the number of work requests by 12%. The later is corroborated by our simulation that shows a decrease on the number of work requests between 10% and 15%.

5.1. **Improving the analysis by changing the potential function.** We consider the same model of unitary tasks as in Section 4. The potential function of our system is defined as

$$\Phi_t = \sum_{i=1}^m w_i(t)^\nu,$$

where $\nu > 1$ is a constant factor.

When an idle processor steals a processor with $w_i(t)$ tasks, the potential decreases by

$$\delta_i = w_i(t)^\nu - \left\lceil \frac{w_i(t) - 1}{2} \right\rceil^\nu + \left\lfloor \frac{w_i(t) - 1}{2} \right\rfloor^\nu \geq w_i(t)^\nu - \left\lfloor \frac{w_i(t)}{2} \right\rfloor^\nu + \left\lfloor \frac{w_i(t)}{2} \right\rfloor^\nu$$
$$\geq \left(1 - 2^{1-\nu}\right) w_i(t)^\nu.$$

This shows that the expected value of the potential at time $t + 1$ is

$$\mathbb{E}\left[\Phi_{t+1}\right] \leq (1 - q(r)(1 - 2^{1-\nu})) \cdot \Phi_t.$$

where $q(r)$ is the probability for a processor to receive at least one work request when r processors are stealing, $q(r) = 1 - \left(1 - \frac{1}{m-1}\right)^r$.

Following the analysis of the previous part, and as $\Phi_0 \leq W^\nu$ the expected makespan is bounded by:

$$\frac{W}{m} + \lambda(\nu) \cdot \left(\log \Phi_0 + 1 + \frac{1}{\ln 2}\right) \leq \frac{W}{m} + \nu\lambda(\nu) \cdot \left(\log W + 1 + \frac{1}{\ln 2}\right),$$

where $\lambda(\nu)$ is a constant depending on ν equal to:

$$(8) \qquad \lambda(\nu) \stackrel{\text{def}}{=} \max_r \left\{ \frac{r}{-\log_2(1 - q(r)(1 - 2^{1-\nu}))} \right\}$$

As for $\nu = 2$ of Section 4, it can be shown the maximum of Equation 8 is attained for $r = m - 1$.

The constant factor in front of $\log W$ is $\nu\lambda(\nu)$. Numerically, the minimum of $\nu\lambda(\nu)$ is for $\nu \approx 2.94$ and is less than 3.24.

Theorem 3. *Let $C_{\max}$ be the makespan of $W = n$ unit independent tasks scheduled DLS. Then:*

$$\mathbb{E}\left[C_{\max}\right] \leq \frac{W}{m} + 3.24 \cdot \left(\log_2 W + \frac{1}{2\ln 2}\right) + 1$$

In Section 4, we have shown that the makespan was bounded by

$$\frac{W}{m} + 2\lambda(2) \cdot \left(\log_2 \Phi_0 + \frac{1}{\ln 2}\right) + 1 \leq \frac{W}{m} + 3.65 \cdot \left(\log_2 W + \frac{1}{2\ln 2}\right) + 1.$$

Theorem 3 improves the constant factor in front of $\log_2 W$. However, we loose the information of the initial repartition of tasks Φ_0.

5.2. **Cooperation among thieves.** In this section, we modify the protocol for managing the distributed list. Previously, when $k > 1$ steal requests were sent on the same processor, only one of them could be served due to contention on the list. We now allow the k requests to be served in unit time. This model has been implemented in the middleware Kaapi (Gautier et al, 2007). When k steal requests target the same processor, the work is divided into $k + 1$ pieces. In practice, allowing concurrent thieves increase the cost of a steal request but we neglect this additional cost here. We assume that the k concurrent steal requests can be served in unit time. We study the influence of this new protocol on the number of steal requests in the case of unit independent tasks.

We define the potential of the system at time t to be:

$$\Phi(t) = \sum_{i=1}^{m} \left(w_i(t)^\nu - w_i(t)\right).$$

Let us first compute the decrease of the potential when processor i receives $k \geq 1$ steal requests. If $w_i(t) > 0$, it can be written $w_i(t) = (k + 1)q + b$ with $0 \leq b < k + 1$. We neglect the decrease of potential due to the execution tasks ($\nu > 1$ implies that execution of tasks decreases the potential).

After one time step and k steal requests, the work will be divided into r parts with $q + 1$ tasks and $k + 1 - r$ parts with q tasks. $\sum_i w_i(t)$ does not vary during the stealing phase. Therefore, the difference of potential due to these k work requests is

$$\delta_i^k = ((k + 1)q + b)^\nu - b(q + 1)^\nu - (k + 1 - b)q^\nu.$$

Let us denote $\alpha \overset{\text{def}}{=} b/(k+1) \in [0; 1)$ and let $f(x) = (x+\alpha)^\nu + (1 - 2^{1-\nu})(x+\alpha) - (1 - \alpha)x^\nu - \alpha(x+1)^\nu$. The first derivative of f is $f'(x) = \nu(x+\alpha)^{\nu-1} + (1 - 2^{1-\nu}) - \nu(1 - \alpha)x^{\nu-1} - \alpha(x+1)^{\nu-1}$ and the derivative of f' is $f''(x) = \nu(1 - \nu)((x+\alpha)^{\nu-2} - (1 - \alpha)x^{\nu-2} - \alpha(x+1)^{\nu-2}$. As $\nu < 3$, the function $x \mapsto x^{\nu-2}$ is concave which implies than $f''(x) \geq 0$. Therefore, f' is increasing. Moreover, $f'(0) = \nu(\alpha^{\nu-1} - \alpha) + (1 - 2^{1-\nu}) \geq 0$. This shows that for all x, $f'(x) \geq 0$ and that f is increasing. The value of f in 0 is $f(0) = \alpha^\nu - (1 - 2^{1-\nu})\alpha - \alpha = \alpha^\nu(1 - (2\alpha)^{1-\nu}) \geq 0$ which implies that for all x, $f(x) \geq 0$.

Recall that $w_i(t) = (k + 1)q + b$ and $\alpha = b/(k + 1)$. Using the notation f and the fact that $(k + 1)^{1-\nu} \leq 2^{1-\nu}$, the decrease of potential δ_i^k can be written

$$\delta_i^k = (1 - (k + 1)^{1-\nu}) \cdot (w_i(t)^\nu - w_i(t)) + (k + 1) \cdot f(q)$$

$$(9) \qquad \geq (1 - (k + 1)^{1-\nu}) \cdot (w_i(t)^\nu - w_i(t)).$$

Let $q_k(r)$ be the probability for a processor to receive k work requests when r processors are stealing. $q_k(r)$ is equal to:

$$q_k(r) = \binom{r}{k} \frac{1}{(m - 1)^k} \left(\frac{m - 2}{m - 1}\right)^{r-k}$$

The expected decrease of the potential caused by the steals on processor i is equal to $\sum_{k=0}^{r} \delta_i^k q_k(r)$. Using equation (9), we can bound the expected potential at time $t+1$ by

$$\mathbb{E}\left[\Phi_t - \Phi_{t+1} \mid \mathscr{F}_t\right] = \sum_{i=0}^{m} \sum_{k=0}^{r} \delta_i^k \cdot q_k(r)$$

$$\mathbb{E}\left[\Phi_{t+1} \mid \mathscr{F}_t\right] \leq \left(1 - \sum_{k=0}^{r}(1 - (k+1)^{1-\nu}) \cdot q_k(r)\right) \cdot \Phi_t$$

Theorem 4. *The makespan $C_{\max}^{\text{coop}}$ of $W = n$ unit independent tasks scheduled with cooperative work stealing satisfies:*

(i) $\mathbb{E}\left[C_{\max}^{\text{coop}}\right] \leq \dfrac{W}{m} + 2.88 \cdot \log_2 W + 3.4$

(ii) $\mathbb{P}\left\{C_{\max}^{\text{coop}} \geq \dfrac{W}{m} + 2.88 \cdot \log_2 W + 2 + \log_2\left(\dfrac{1}{\epsilon}\right)\right\} \leq \epsilon.$

Proof. The proof is very similar to the one of Theorem 2. Let

$$h(r) \stackrel{\text{def}}{=} 1 - \sum_{k=0}^{r}(1 - (k+1)^{1-\nu}) \cdot q_k(r)$$

and

$$\lambda^{\text{coop}}(\nu) \stackrel{\text{def}}{=} \max_{1 \leq r \leq m} \frac{r}{-m \cdot \log_2 h(r)}.$$

Using Theorem 1, we have:

$$\mathbb{E}\left[C_{\max}^{\text{coop}}\right] \leq \frac{W}{m} + \nu \lambda^{\text{coop}}(\nu) \cdot \log_2 W + \frac{\lambda(\nu)}{\ln 2} + 1.$$

In the general case the exact computation of $h(r)$ is intractable. However, by a numerical computation, one can show that $3\lambda^{\text{coop}}(3) < 2.88$.

When $\Phi_t < 1$, we have $\sum_i w_i(t)^\nu - w_i(t) < 1$. This implies that for all processor i, $w_i(t)$ equals 0 or 1. This adds (at most) one step of computation at the end of the schedule. As $\lambda(3)/\ln(2) + 1 + 1 = 3.4$, we obtain the calimed bound. $\square$ $\square$

Compared to the situation with no cooperation among thieves, the number of steal requests is reduced by a factor $3.24/2.88 \approx 12\%$. We will see in Section 8 that this is close to the value obtained by simulation.

Remark. The exact computation can be accomplished for $\nu = 2$ (Tchiboukdjian et al, 2010) and leads to a constant factor of $2\lambda^{\text{coop}}(2) \leq -2/\log_2(1 - \frac{1}{e}) < 3.02$.

6. Weighted independent tasks

In this section, we analyze the number of work requests for weighted independent tasks. Each task j has a processing time p_j which is unknown. When an idle processor attempts to steal a processor, half of the tasks of the victim are transfered from the active processor to the idle one. A task that is currently executed by a processor cannot be stolen. If the victim has $2k(+1)$ tasks (plus one for the task that is currently executed), the work is split in $k(+1)$, k. If the victim has $2k + 1(+1)$ tasks, the work is split in $k(+1)$, $k + 1$.

In all this analysis, we consider that the scheduler does not know the weight of the different tasks p_j. Therefore, when the work is split in two parts, we do not assume that the work is split fairly (see for example Figure 3) but only that the number of tasks is split in two equal parts.
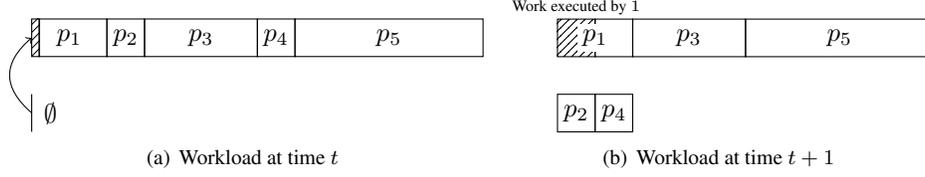
(a) Workload at time t (b) Workload at time $t + 1$

FIGURE 3. Evolution of the repartition of tasks during one time step. At time t, one processor has all the tasks. p_1 can not be stolen since the processor 1 has already started executing it. After one work request done by the second processor, one processor has 3 tasks and one has 2 tasks but the workload may be very different, depending on the processing times p_j.

6.1. **Definition of the potential function and expected decrease.** As the processing times are unknown, the work cannot be shared evenly between both processors and can be as bad as one processor getting all the smallest tasks and one all the biggest tasks (see Figure 3). Let us call $w_i(t)$ the *number of tasks* possessed by the processor i. The potential of the system at time t is defined as:

$$(10) \qquad \Phi_t \stackrel{\text{def}}{=} \sum_i \left(w_i(t)^\nu - w_i(t) \right).$$

During a work request, half of the tasks are transfered from an active processor to the idle processor. If the processor j is stealing tasks from processor i, the number of tasks possessed by i and j at time $t+1$ are $w_j(t+1) = \left\lceil w_i(t)/2 \right\rceil$ and $w_i(t+1) = \left\lfloor w_i(t)/2 \right\rfloor$. Therefore, the decrease of potential is equal to the one of the cooperative steal of Equation 9 for $k = 1$:

$$\delta_i \geq (1 - 2^{1-\nu}) \cdot \left(w_i(t)^\nu - w_i(t) \right).$$

Following the analysis of Section 5.2, this shows that in average:

$$(11) \qquad \mathbb{E}\left[\Phi_{t+1}\right] \leq \left(1 - (1 - 2^{1-\nu})q(r)\right) \cdot \Phi_t.$$

6.2. **Bound on the makespan.** Equation 11 allows us to apply Theorem 1 to derive a bound on the makespan of weighted tasks by the distributed list scheduling algorithm. This bound differs from the one for unit tasks only by an additive term of $p_{\max}$.

Theorem 5. *Let* $p_{\max} \stackrel{\text{def}}{=} \max p_j$ *be the maximum processing times. The expected makespan to schedule n weighted tasks of total processing time $W = \sum p_j$ by DLS is bounded by*

$$\mathbb{E}\left[C_{\max}\right] \leq \frac{W}{m} + \frac{m-1}{m} p_{\max} + 3.24 \cdot \left(\log_2 n + \frac{1}{2 \ln 2} \right) + 1$$

Proof. Let Φ_t be the potential defined by Equation 10. At time $t = 0$, the potential of the system is bounded by $W^\nu - W$. Therefore, by Theorem 1, the number of work requests before $\Phi_t < 1$ is bounded by

$$m \cdot \lambda \cdot \left(\log_2 \Phi_0 + 1 + \frac{1}{\ln 2} \right) \leq m \cdot \nu\lambda(\nu) \cdot \left(2 \log_2 W + 1 + \frac{1}{\ln 2} \right),$$

where $\nu\lambda(\nu) < 3.24$ is the same constant as the bound for the unit tasks with the potential function $\sum_i w_i^\nu$ of Theorem 3.

As $\Phi_t \in \mathbb{N}$, $\Phi_t < 1$ implies that $\Phi_t = 0$. Moreover, by definition of Φ_t, this implies that for all i: $w_i(t)^\nu - w_i(t) = 0$, which implies that for all i: $w_i(t) \leq 1$. Therefore, once Φ_t is equal to 0, there is at most one task per processor. This phase can last for at most $p_{\max}$ unit of time, generating at most $(m-1)p_{\max}$ work requests. □ □

Remark. The same analysis applies for the cooperative stealing scheme of Section 5.2 leading to the same improved bound in $2.88 \log_2 n$ instead of $3.24 \log_2 n$.

7. TASKS WITH PRECEDENCES

In this section, we show how the well known non-blocking work stealing of Arora et al (2001) (denoted ABP in the sequel) can be analyzed with our method which provides tighter bounds for the makespan. We first recall the WS scheduler of ABP, then we show how to define the amount of work on a processor $w_i(t)$, finally we apply the analysis of Section 3 to bound the makespan.

7.1. **ABP work-stealing scheduler.** Following Arora et al (2001), a multithreaded computation is modeled as a directed acyclic graph G with W unit tasks task and edges define precedence constraints. There is a single source task and the out-degree is at most 2. The critical path of G is denoted by D. ABP schedules the DAG G as follows. Each processor i maintains a double-ended queue (called a deque) Q_i of ready tasks. At each slot, an active processor i with a non-empty deque executes the task at the bottom of its deque Q_i; once its execution is completed, this task is popped from the bottom of the deque, enabling – *i.e.* making ready – 0, 1 or 2 child tasks that are pushed at the bottom of Q_i. At each top, an idle processor j with an empty deque Q_j becomes a thief: it performs a steal request on another randomly chosen victim deque; if the victim deque contains ready tasks, then its top-most task is popped and pushed into the deque of one of its concurrent thieves. If j becomes active just after its steal request, the steal request is said successful. Otherwise, Q_j remains empty and the steal request fails which may occur in the three following situations: either the victim deque Q_i is empty; or, Q_i contains only one task currently in execution on i; or, due to contention, another thief performs a successful steal request on i simultaneously.

7.2. **Definition of $w_i(t)$.** Let us first recall the definition of the *enabling tree* of Arora et al (2001). If the execution of task u enables task v, then the edge (u,v) of G is an enabling edge. The sub-graph of G consisting of only enabling edges forms a rooted tree called the enabling tree. We denote by $h(u)$ the height of a task u in the enabling tree. The root of the DAG has height D. Moreover, it has been shown in Arora et al (2001) that tasks in the deque have strictly decreasing height from top to bottom except for the two bottom most tasks which can have equal heights.

We now define $w_i(t)$, the amount of work on processor i at time t. Let h_t be the maximum height of all tasks in the deque. If the deque contains at least two tasks including the one currently executing we define $w_i(t) = (2\sqrt{2})^{h_t}$. If the deque contains only one task currently executing we define $w_i(t) = \frac{1}{2} \cdot (2\sqrt{2})^{h_t}$. The following lemma states that this definition of $w_i(t)$ behaves in a similar way than the one used for the independent unit tasks analysis of Section 4.

Lemma 1. *For any active processor i, we have $w_i(t+1) \leq w_i(t)$. Moreover, after any successful steal request from a processor j on i, $w_i(t+1) \leq w_i(t)/2$ and $w_j(t+1) \leq w_i(t)/2$ and if all steal requests are unsuccessful we have $w_i(t+1) \leq w_i(t)/\sqrt{2}$.*

Proof. We first analyze the execution of one task u at the bottom of the deque. Executing task u enables at most two tasks and these tasks are the children of u in the enabling tree. If the deque contains more than one task, the top most task has height h_t and this task is still in the deque at time $t + 1$. Thus the maximum height does not change and $w_i(t) = w_i(t + 1)$. If the deque contains only one task, we have $w_i(t) = \frac{1}{2} \cdot (2\sqrt{2})^{h_t}$ and $w_i(t + 1) \leq (2\sqrt{2})^{h_t-1}$. Thus $w_i(t + 1) \leq w_i(t)$.

We now analyze a successful steal from processor j. In this case, the deque of processor i contains at least two tasks and $w_i(t) = (2\sqrt{2})^{h_t}$. The stolen task is one with the maximum height and is the only task in the deque of processor j thus $w_j(t+1) = \frac{1}{2} \cdot (2\sqrt{2})^{h_t} \leq w_i(t)/2$. For the processor i, either its deque contains only one task after the steal with height at most h_t and $w_i(t + 1) \leq \frac{1}{2} \cdot (2\sqrt{2})^{h_t} \leq w_i(t)/2$, either there are still more than 2 tasks and $w_i(t + 1) \leq (2\sqrt{2})^{h_t-1} < w_i(t)/2$.

Finally, if all steal requests are unsuccessful, the deque of processor i contains at most one task. If the deque is empty $w_i(t + 1) = w_i(t) = 0$ and thus $w_i(t + 1) \leq w_i(t)/\sqrt{2}$. If the deque contains exactly one task, $w_i(t) = \frac{1}{2} \cdot (2\sqrt{2})^{h_t}$ and $w_i(t + 1) \leq (2\sqrt{2})^{h_t-1}$ thus $w_i(t + 1) \leq w_i(t)/\sqrt{2}$. $\qquad\square$ $\qquad\square$

7.3. **Bound on the makespan.** To study the number of steals, we follow the analysis presented in Section 3 with the potential function $\Phi(t) = \sum_i w_i(t)^2$. Using results from lemma 1, we compute the decrease of the potential $\delta_i(t)$ due to steal requests on processor i by distinguishing two cases. If there is a successful steal from processor j,

$$\delta_i(t) = w_i(t)^2 - w_i(t + 1)^2 - w_j(t + 1)^2 \geq w_i(t)^2 - 2 \cdot \left(\frac{w_i(t)}{2}\right)^2 \geq \frac{1}{2} \cdot w_i(t)^2.$$

If all steals are unsuccessful, the decrease of the potential is

$$\delta_i(t) = w_i(t)^2 - w_i(t + 1)^2 \geq w_i(t)^2 - \left(\frac{w_i(t)}{\sqrt{2}}\right)^2 \geq \frac{1}{2} \cdot w_i(t)^2.$$

In all cases, $\delta_i(t) \geq w_i(t)^2/2$. We obtain the expected potential at time $t + 1$ by summing the expected decrease on each active processor:

$$\mathbb{E}\left[\Phi_t - \Phi_{t+1} \mid \mathscr{F}_t\right] \geq \sum_{i=0}^{m} \frac{w_i(t)^2}{2} \cdot q(r_t)$$

$$\mathbb{E}\left[\Phi_{t+1} \mid \mathscr{F}_t\right] \leq \left(1 - \frac{q(r_t)}{2}\right) \cdot \Phi(t)$$

Finally, we can state the following theorem.

Theorem 6. *On a DAG composed of W unit tasks, with critical path D, one source and out-degree at most 2, the makespan of ABP work stealing verifies:*

(i) $\mathbb{E}\left[C_{\max}\right] \leq \dfrac{W}{m} + \dfrac{3}{1 - \log_2(1 + \frac{1}{e})} \cdot D + 1 < \dfrac{W}{m} + 5.5 \cdot D + 1.$

(ii) $\mathbb{P}\left\{C_{\max} \geq \dfrac{W}{m} + \dfrac{3}{1 - \log_2(1 + \frac{1}{e})} \cdot \left(D + \log_2 \frac{1}{\epsilon}\right) + 1\right\} \leq \epsilon$

Proof. The proof is a direct application of Theorem 1. As in the initial step there is only one non empty deque containing the root task with height D, the initial potential is

$$\Phi(0) = \left(\frac{1}{2} \cdot \left(2\sqrt{2}\right)^{D}\right)^2.$$

Thus the expected number of steal requests before $\Phi(t) < 1$ is bounded by

$$\mathbb{E}\left[R\right] \leq \lambda \cdot m \cdot \log_2\left[\left(\frac{1}{2} \cdot \left(2\sqrt{2}\right)^D\right)^2\right] + m \cdot \left(1 + \frac{\lambda}{\ln(2)}\right)$$

$$\leq 2\lambda \cdot m \cdot D \cdot \log_2(2\sqrt{2}) + m \cdot \left(1 + \frac{\lambda}{\ln(2)} - 2\lambda\right)$$

$$\leq 3\lambda \cdot m \cdot D \qquad\qquad (\text{as } 1 + \lambda/\ln(2) - 2\lambda < 0)$$

where $\lambda = (1 - \log_2(1 + 1/e))^{-1}$ is the same constant as the bound for the unit tasks of Section 4.

Moreover, when $\Phi(t) < 1$, we have $\forall i, w_i(t) < 1$. There is at most one task of height 0 in each deque, *i.e.* a leaf of the enabling tree which cannot enable any other task. This last step generates at most $m - 1$ additional steal requests. In total, the expected number of steal requests is bounded by $\mathbb{E}\left[R\right] \leq 3\lambda \cdot m \cdot D + m - 1$. The bound on the makespan is obtained using the relation $m \cdot C_{\max} = W + R$.

The proof of (i) applies *mutatis mutandis* to prove the bound in probability (ii). □ □

Remark. In Arora et al (2001), the authors established the upper bounds :

$$\mathbb{E}\left[C_{\max}\right] \leq \frac{W}{m} + 32 \cdot D \text{ and } \mathbb{P}\left\{C_{\max} \geq \frac{W}{m} + 64 \cdot D + 16 \cdot \log_2 \frac{1}{\epsilon}\right\} \leq \epsilon$$

in Section 4.3, proof of Theorem 9. Our bounds greatly improve the constant factors of this previous result.

8. EXPERIMENTAL STUDY

The theoretical analysis gives an upper bounds on the expected value of the makespan and deviation from the mean for the various models we considered. In this section, we study experimentally the distribution of the makespan. Statistical tests give evidence that the makespan for independent tasks follows a generalized extreme value (gev) distribution (Kotz and Nadarajah, 2001). This was expected since such a distribution arises when dealing with maximum of random variables. For tasks with dependencies, it depends on the structure of the graph: DAGs with short critical path still follow a gev distribution but when the critical path grows, it tends to a gaussian distribution. We also study in more details the overhead to W/m and show that it is approximately $2.37 \log_2 W$ for unit independent tasks which is close to the theoretical result of $3.24 \log_2 W$ (*cf.* Section 5).

We developed a simulator that strictly follows our model. At the beginning, all the tasks are given to processor 0 in order to be in the worst case, *i.e.* when the initial potential Φ_0 is maximum. Each pair (m, W) is simulated 10000 to get accurate results, with a coefficient of variation about 2%.

8.1. **Distribution of the makespan.** We consider here a fixed workload $W = 2^{17}$ on $m = 2^{10}$ processors for independent tasks and $m = 2^7$ processors for tasks with dependencies. For the weighted model, processing times were generated randomly and uniformly between 1 and 10. For the DAG model, graphs have been generated using a layer by layer method. We generated two types of DAGs, one with a short critical path (close to the minimum possible $\log_2 W$) and the other one with a long critical path (around $W/4m$ in order to keep enough tasks per processor per layer). Fig. 4 presents histograms for $C_{\max} - \lceil W/m \rceil$.

The distributions of the first three models (a,b,c in Fig. 4) are clearly not gaussian: they are asymmetrical with an heavier right tail. To fit these three models, we use the generalized extreme value (gev) distribution (Kotz and Nadarajah, 2001). In the same way as the

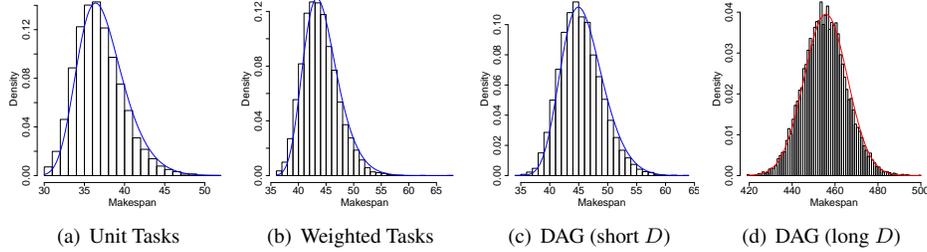(a) Unit Tasks   (b) Weighted Tasks   (c) DAG (short D)   (d) DAG (long D)

FIGURE 4. Distribution of the makespan for unit independent tasks 4(a), weighted independent tasks 4(b) and tasks with dependencies 4(c) and 4(d). The first three models follow a gev distribution (blue curves), the last one is gaussian (red curve).

normal distribution arises when studying the sum of independent and identically distributed (iid) random variables, the gev distribution arises when studying the maximum of iid random variables. The extreme value theorem, an equivalent of the central limit theorem for maxima, states that the maximum of iid random variables converges in distribution to a gev distribution. In our setting, the random variables measuring the load of each processor are not independent, thus the extreme value theorem cannot apply directly. However, it is possible to fit the distribution of the makespan to a gev distribution. In Fig. 4, the fitted distributions (blue curve) closely follow the histograms. To confirm this graphical approach, we performed a goodness of fit test. The χ^2 test is well-suited to our data because the distribution of the makespan is discrete. We compared the results of the best fitted gev to the best fitted gaussian. The χ^2 test strongly rejects the gaussian hypothesis but does not reject the gev hypothesis with a p-value of more than 0.5. This confirms that the makespan follows a gev distribution. We fitted the last model, DAG with long critical path, with a gaussian (red curve in Fig. 4(d)). In this last case, the completion time of each layer of the DAG should correspond to a gev distribution but the total makespan, the sums of all layers, should tend to a gaussian by the central limit theorem. Indeed the χ^2 test does not reject the gaussian hypothesis with a p-value around 0.3.

8.2. **Study of the** $\log_2 W$ **term.** We focus now on unit independent tasks as the other models rely on too many parameters (the choice of the processing times for weighted tasks and the structure of the DAG for tasks with dependencies). We want to show that the number of work requests is proportional to $\log_2 W$ and study the proportionality constant. We first launch simulations with a fixed number of processors m and a wide range of work in successive powers of 10. A linear regression confirms the linear dependency in $\log_2 W$ with a coefficient of determination ("r squared") greater than 0.9999[1].

Then, we obtain the slope of the regression for various number of processors. The value of the slope tends to a limit around 2.37 (*cf.* Fig. 5(left)). This shows that the theoretical analysis of Theorem 2 is almost accurate with a constant of approximately 3.24. We also study the constant factor of $\log_2 W$ for the cooperative steal of Section 5. The theoretical value of 2.88 is again close to the value obtained by simulation 2.08 (*cf.* Figure 5(left)). The difference between the theoretical and the practical values can be explained by the worst case analysis on the number of steal requests per time step in Theorem 1.
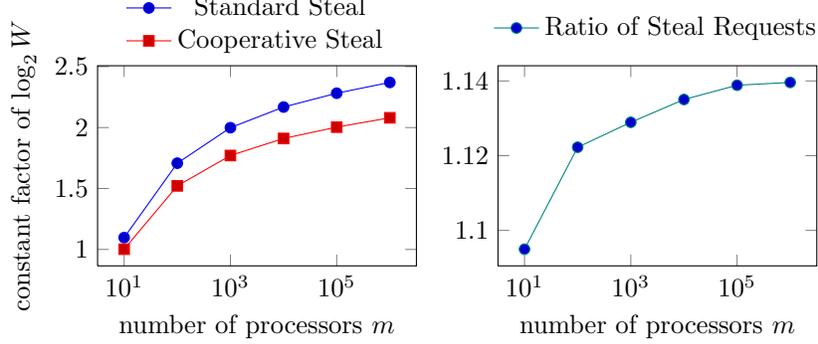
---

[1]the closer to 1, the better

FIGURE 5. (Left) Constant factor of $\log_2 W$ against the number of processors for the standard steal and the cooperative steal. (Right) Ratio of steal requests (standard/cooperative).

Moreover, simulations in Fig. 5(right) show that the ratio of steal requests between standard and cooperative steals goes asymptotically to 14%. The ratio between the two corresponding theoretical bounds is about 12%. This indicates that the biais introduced by our analysis is systematic and thus, our analysis may be used as a good prediction while using cooperation among thieves.

9. CONCLUDING REMARKS

In this paper, we presented a complete analysis of the cost of distribution in list scheduling. We proposed a new framework, based on potential functions, for analyzing the complexity of distributed list scheduling algorithms. In all variants of the problem, we succeeded to characterize precisely the overhead due to the decentralization of the list. These results are summarized in the following table comparing makespans for standard (centralized) and decentralized list scheduling.

	Centralized	Decentralized (WS)
Unit Tasks ($W = n$)	$\left\lceil \dfrac{W}{m} \right\rceil$	$\dfrac{W}{m} + 3.24 \log_2 W + 3.33$
Initial repartition	–	$\dfrac{W}{m} + 1.83 \log_2 \displaystyle\sum_{i=0}^{m} \left(w_i - \dfrac{W}{m} \right)^2 + 3.63$
Cooperative	–	$\dfrac{W}{m} + 2.88 \log_2 W + 3.4$
Weighted Tasks	$\dfrac{W}{m} + \dfrac{m-1}{m} \cdot p_{\max}$	$\dfrac{W}{m} + \dfrac{m-1}{m} \cdot p_{\max} + 3.24 \log_2 n + 3.33$
Tasks w. predecences	$\dfrac{W}{m} + \dfrac{m-1}{m} \cdot D$	$\dfrac{W}{m} + 5.5D + 1$

In particular, in the case of independent tasks, the overhead due to the distribution is small and only depends on the number of tasks and not on their weights. In addition, this analysis improves the bounds for the classical work stealing algorithm of Arora et al (2001)

from $32D$ to $5.5D$. We believe that this work should help to clarify the links between classical list scheduling and work stealing.

Furthermore, the framework to analyze DLS algorithms described in this paper is more general than the method of Arora et al (2001). Indeed, we do not assume a specific rule (*e.g.* depth first execution of tasks) to manage the local lists. Moreover, we do not refer to the structure of the DAG (*e.g.* the depth of a task in the enabling tree) but on the work contained in each list. Thus, we plan to extend this analysis to the case of general precedence graphs.

REFERENCES

Adler M, Chakrabarti S, Mitzenmacher M, Rasmussen L (1995) Parallel randomized load balancing. In: Proceedings of STOC, pp 238–247

Arora NS, Blumofe RD, Plaxton CG (2001) Thread scheduling for multiprogrammed multiprocessors. Theory of Computing Systems 34(2):115–144

Azar Y, Broder AZ, Karlin AR, Upfal E (1999) Balanced allocations. SIAM Journal on Computing 29(1):180–200, DOI 10.1137/S0097539795288490

Bender MA, Rabin MO (2002) Online scheduling of parallel programs on heterogeneous systems with applications to cilk. Theory of Computing Systems 35:2002

Berenbrink P, Friedetzky T, Goldberg LA (2003) The natural work-stealing algorithm is stable. SIAM Journal of Computing 32(5):1260–1279

Berenbrink P, Friedetzky T, Goldberg LA, Goldberg PW, Hu Z, Martin R (2007) Distributed selfish load balancing. SIAM Journal on Computing 37(4), DOI 10.1137/060660345

Berenbrink P, Friedetzky T, Hu Z, Martin R (2008) On weighted balls-into-bins games. Theoretical Computer Science 409(3):511 – 520

Berenbrink P, Friedetzky T, Hu Z (2009) A new analytical method for parallel, diffusion-type load balancing. Journal of Parallel and Distributed Computing 69(1):54 – 61

Blumofe RD, Leiserson CE (1999) Scheduling multithreaded computations by work stealing. Journal of the ACM 46(5):720–748

Chekuri C, Bender M (2001) An efficient approximation algorithm for minimizing makespan on uniformly related machines. Journal of Algorithms 41(2):212 – 224

Drozdowski M (2009) Scheduling for Parallel Processing. Springer

Frigo M, Leiserson CE, Randall KH (1998) The implementation of the Cilk-5 multithreaded language. In: Proceedings of PLDI

Gast N, Gaujal B (2010) A Mean Field Model of Work Stealing in Large-Scale Systems. In: Proceedings of SIGMETRICS

Gautier T, Besseron X, Pigeon L (2007) KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In: Proceedings of PASCO, pp 15–23

Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17:416–429

Hwang JJ, Chow YC, Anger FD, Lee CY (1989) Scheduling precedence graphs in systems with interprocessor communication times. SIAM Journal on Computing 18(2):244–257

Kotz S, Nadarajah S (2001) Extreme Value Distributions: Theory and Applications. World Scientific Publishing Company

Leung J (2004) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press

Mitzenmacher M (1998) Analyses of load stealing models based on differential equations. In: Proceedings of SPAA, pp 212–221

Robert Y, Vivien F (2009) Introduction to Scheduling. Chapman & Hall/CRC Press

Robison A, Voss M, Kukanov A (2008) Optimization via reflection on work stealing in TBB. In: Proceedings of IPDPS, pp 1–8

Schwiegelshohn U, Tchernykh A, Yahyapour R (2008) Online scheduling in grids. In: Proceedings of IPDPS

Tchiboukdjian M, Gast N, Trystram D, Roch JL, Bernard J (2010) A tighter analysis of work stealing. In: The 21st International Symposium on Algorithms and Computation (ISAAC)

Traoré D, Roch JL, Maillard N, Gautier T, Bernard J (2008) Deque-free work-optimal parallel STL algorithms. In: Proceedings of Euro-Par, pp 887–897

MARC TCHIBOUKDJIAN CNRS / CEA,DAM,DIF - ARPAJON, FRANCE
E-mail address: `marc.tchiboukdjian@imag.fr`

NICOLAS GAST, EPFL, IC-LCA2, BÂTIMENT BC, STATION 14, 1015 LAUSANNE, SWITZERLAND
E-mail address: `nicolas.gast@epfl.ch`

DENIS TRYSTRAM GRENOBLE UNIVERSITY, 51 AV JEAN-KUNTZMAN, 38330 MONTBONNOT, FRANCE
E-mail address: `denis.trystram@imag.fr`