A Tighter Analysis of Work Stealing

Marc Tchiboukdjian[1], Nicolas Gast[1], Denis Trystram[1], Jean-Louis Roch[1], and Julien Bernard[2]

[1] Grenoble University
firstname.lastname@imag.fr
[2] Université de Franche-Comté
julien.bernard@lifc.univ-fcomte.fr

Abstract. Classical list scheduling is a very popular and efficient technique for scheduling jobs in parallel platforms. However, with the increasing number of processors, the cost for managing a single centralized list becomes prohibitive. The objective of this work is to study the extra cost that must be paid when the list is distributed among the processors. We 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. A bound on the deviation from the mean is also derived. Then, we apply this technique to show that the expected makespan for scheduling W unit independent tasks on m processors is equal to W/m with an additional term in $3.65 \log_2 W$. Finally, we analyze the work stealing algorithm of Arora, Blumofe and Plaxton and significantly improve the bound on the number of steals. Moreover, simulations show that our bound is very close to the exact value, approximately 50% off. This new analysis also enables to study the influence of the initial repartition of tasks and the reduction of the number of steals when several thieves can simultaneously steal work in the same processor's list.

1 Introduction

List scheduling is one of the most popular technique for scheduling the tasks of a parallel program. This algorithm has been introduced by Graham [1]. Its principle is to build a list of ready tasks and schedule them as soon as there exist available resources. List schedules are low-cost (greedy) algorithms that are not too far from optimal solutions. Most proposed list algorithms always consider a centralized management of the list. However, today parallel platforms involve more and more processors. Thus, the time needed for managing such a centralized data structure can not be ignored anymore [2]. Practically, implementing such schedulers induces synchronization overheads when several processors access the list concurrently. Such overheads involve low level synchronization mechanisms.

A suitable approach to reduce the contention is to distribute the list among the processors: each processor manages its own list of tasks. When a processor becomes idle, it randomly chooses another processor and steals some work, *i.e.* it transfers some tasks from the victim's list to its own list. Such a strategy is called *work-stealing* (WS). WS has been implemented in many languages and parallel libraries including Cilk [3], Intel TBB [4] and KAAPI [5].

Related works. WS has been analyzed in a seminal paper of Blumofe and Leiserson [6] where they show that the expected makespan of series-parallel precedence graphs with unit tasks is bounded by $\mathbb{E}\left[C_{\max}\right] \leq W/m + \mathrm{O}(T_\infty)$ where W is the number of tasks, T_∞ is the critical path of the graph and m is the number of processors. This analysis has been improved in [7] using a proof based on a potential function. However the precedence graph is constrained to have only one source and out-degree at most 2 which does not model the basic case of independent tasks. Simulating independent tasks with a binary tree of dependencies gives a bound of $W/m + \mathrm{O}(\log W)$ as a complete binary tree of W nodes has a depth of $T_\infty \leq \log_2 W$. Our new analysis allows to directly devise a result for the independent tasks case.

Notice that there exist other ways to analyze work stealing where the work generation is probabilistic and that target steady state results [8–11].

Our analysis shows some similarities with the work of Berenbrink *et al.* [12]. It is based on computing the expected decrease of a potential function. However, to simplify the analysis, we introduce an adversary that controls one parameter of the model, the number of steal requests at each time step.

Contributions. We present a new methodology for studying distributed list scheduling algorithms. Based on the analysis of the load balancing between two processors during a steal request, the expected number of steals can be deduced and a bound on the makespan is derived. The methodology is generic and it is applied to the case of independent tasks and to the WS algorithm of [7]. Our new analysis improves the classical bound on the number of steal requests of WS for scheduling precedence graphs. Constant factors are greatly reduced and are less than 50% away from values obtained by simulation. Moreover, our analysis enables to evaluate precisely the impact of several modifications of the WS.

Roadmap. After presenting the model and notations in Section 2, we give the principle of the analysis in Section 3. We apply this analysis to the case of unit independent tasks and study the influence of the initial repartition of tasks in Section 4. Then we extend the analysis for tasks with dependencies in Section 5. Section 6 quantifies the reduction of steals when several thieves are allowed to steal the same victim simultaneously. We analyze simulation results in Section 7.

2 Model of the Distributed List

In this section, we give properties a distributed list implementation should follow.

We consider a parallel platform composed of m identical processors. At time t, let $w_i(t)$ represent the amount of work on processor i. When $w_i(t) > 0$, processor i is active and executes some work: $w_i(t+1) \leq w_i(t)$. When $w_i(t) = 0$, processor i is idle and intends to steal a random processor j. If processor j has no work, *i.e.* $w_j(t) = 0$, the steal fails and processor i will steal again at the next time slot. Otherwise, a certain amount of work is transfered from processor j to processor i: $w_i(t+1) + w_j(t+1) \leq w_j(t)$. Processor i will resume execution at time $t+1$. The execution terminates when all the processors are idle, *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 active processors by $\alpha(t) \in [0, m]$. Thus, between time t and $t+1$, there are $m - \alpha(t)$ steal requests. Notice that when $\alpha(t) = 0$, all queues are empty and thus the execution is complete.

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 6.

This is a high level model of a distributed list. We will show in Sections 4 and 5 how these properties accurately model the case of independent tasks and the WS algorithm of [7]. We justify here some choices of this model. There is no explicit communication cost as WS algorithms most often target shared memory platforms. A steal request is done in constant time independently of the amount of work transfered. This assumption is not restrictive: for the case of independent tasks, the description of a large number of tasks can be very short. For instance 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 [13]). For more general cases with dependencies, it is usually enough to transfer a task which represents a part of the graph [7].

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 requests exceeds this bound.

The main idea of our analysis is that we study the decrease of a potential function $\Phi(t)$, instead of studying directly the number of processors that will run out of work and become idle. The definition of $\Phi(t)$ varies depending on the scenario (see Sections 4 to 6). The diminution of the potential depends on the number of steal requests, $m - \alpha(t)$. Since $\alpha(t)$ is a complicated random process, we tackle this problem by assuming that an adversary is choosing the number of active processors $\alpha(t)$ at each step of the schedule. At the beginning, the adversary starts with $\Phi(0)$ potential. Each time, when she chooses its $\alpha(t)$, $m - \alpha(t)$ steal requests are generated and diminishes the potential. The more work requests she creates, the more the potential decreases. She tries to maximize the number of steal requests before running out of potential.

In the actual system, $\alpha(t)$ is determined by the evolution of the system and cannot be chosen at time t. The introduction of an adversary provides an upper-bound on the number of steal requests and has two main advantages. First, its simplicity makes it applicable in several scenarios, such as the ones presented in Sections 4 to 6. Moreover, we show in Section 7 that the gap between the obtained bound and the values obtained by simulation is small.

The analysis of the scenarios of sections 4 to 6 will be done in three steps.

1. First, we define a potential function and we compute the potential decrease $\delta_i^k(t)$ when the processor i receives k work requests.

2. Then we compute the expected decrease of the potential between step t and $t+1$, $\Delta\Phi(t) \overset{\text{def}}{=} \Phi(t) - \Phi(t+1)$. By linearity of expectation,
$$\mathbb{E}\left[\Delta\Phi(t)\right] = \sum_{i=1}^{m}\sum_{k=0}^{m-1} \mathbb{E}\left[\delta_i^k | i \text{ receives } k \text{ requests}\right] \mathbb{P}\left\{i \text{ receives } k \text{ requests}\right\},$$
where $\mathbb{E}[X|Y]$ denotes the expectation of X knowing Y. Using the properties of $\delta_i^k(t)$, we show that there exists a function $h(\alpha) \in (0;1]$ such that
$$\mathbb{E}\left[\Delta\Phi(t) | \Phi(t) = \Phi, \alpha(t) = \alpha\right] \geq h(\alpha)\Phi$$
3. Finally, we obtain a bound on the expected number of steal requests $\mathbb{E}[R]$ using Theorem 1 presented in this section. An upper bound on the expected makespan $\mathbb{E}[C_{\max}]$ can be obtained using the bound on the number of steal.

The following theorem gives an upper bound on the number of steal requests using a lower bound on the expected decrease of the potential in one step.

Theorem 1. *Assume that the potential function $\Phi(t)$ satisfies:*

- *There exists a constant $d > 0$ such that $d\Phi(t) \in \mathbb{N}$.*
- *$\Phi(t)$ is non-increasing.*
- *There exists a function $h(\alpha) \in (0,1]$ such that if $\alpha \in [1, m-1]$ processors are active at time t and $\Phi(t) = \Phi$, then the potential decreases on average by*
$$\mathbb{E}\left[\Phi(t) - \Phi(t+1) | \Phi(t) = \Phi, \alpha(t) = \alpha\right] \geq h(\alpha) \cdot \Phi.$$

Let $\lambda = \max_{1 \leq \alpha \leq m-1} \frac{m-\alpha}{-m \log_2(1-h(\alpha))}$ and $\Phi(0)$ be the potential at $t = 0$. Then

(i) the expected number of steal requests R until $\Phi(t) \leq 1$ is bounded by
$$\mathbb{E}[R] \leq \lambda \cdot m \cdot \log_2 \Phi(0);$$
(ii) The deviation from the mean can be bounded by:
$$\mathbb{P}\left\{R \geq \lambda \cdot m \cdot \log_2 \Phi(0) + u\right\} \leq 2^{-u/(\lambda m)}$$

Proof. Without loss of generality and to simplify the notation, we assume $d = 1$.

Let T be the random variable indicating the end of the schedule: $T = \min\{t | \Phi(t) \leq 1\}$. The number of steal requests is equal to the number of idle processors at each time step. The number of steal requests *after* time t is $R(t)$:
$$R(t) = \sum_{s=t}^{T-1} m - \alpha(s)$$
The total number of steals is $R \overset{\text{def}}{=} R(0)$.

The sequence $\alpha(t)$ is difficult to study since it depends on the number of processors at time $t-1$ with 0 or 1 tasks, but also the successful or unsuccessful steals. Therefore, we perform the analysis assuming a worst-case scenario: at each time t, an adversary can choose $\alpha(t)$ knowing the history of the system but not the future random choices. This can be seen as a Markov decision process with total reward criteria, see [14] for more details about Markov decision processes.

We prove by induction on Φ that for all t,
$$\mathbb{E}\left[R(t) | \Phi(t) = \Phi\right] \leq \lambda m \log_2(\Phi)$$

For $\Phi = 1$, this is clearly true since in that case $T \leq t$ and $R(t) = 0$. Assume that (3) holds for all t and all $\phi < \Phi$. $\mathbb{E}\left[R(t)|\Phi(t) = \Phi\right]$ is equal to:

$$\begin{aligned}\mathbb{E}\left[R(t)|\Phi(t) = \Phi\right] &= \mathbb{E}\left[m - \alpha(t) + R(t+1)|\Phi(t) = \Phi\right] \\ &= m - \alpha(t) + \mathbb{E}\left[R(t+1)|\Phi(t) = \Phi\right] \qquad (1)\end{aligned}$$

By definition of $\Delta\Phi(t)$, if $\Phi(t) = \Phi$, then $\Phi(t+1) = \Phi - \phi$ with probability $\mathbb{P}\left\{\Delta\Phi(t) = \phi|\Phi(t) = \Phi\right\}$. Since $\Phi(t)$ is non-increasing, $\Delta\Phi(t) \geq 0$. Therefore:

$$\mathbb{E}\left[R(t+1)|\Phi(t){=}\Phi\right] = \sum_{\phi=0}^{\Phi} \mathbb{E}\left[R(t+1)|\Phi(t+1){=}\Phi - \phi\right] \mathbb{P}\left\{\Delta\Phi(t) = \phi|\Phi(t){=}\Phi\right\}.$$

Let us denote $p_0 \overset{\text{def}}{=} \mathbb{P}\left\{\Delta\Phi(t) = 0|\Phi(t) = \Phi\right\}$. Using the induction hypothesis, and the fact that $\mathbb{E}\left[R(t+1)|\Phi(t+1) = \Phi\right] = \mathbb{E}\left[R(t)|\Phi(t) = \Phi\right]$, we get from (1)

$$\begin{aligned}(1{-}p_0)\mathbb{E}\left[R(t)|\Phi(t){=}\Phi\right] &\leq m{-}\alpha(t){+}\sum_{\phi=1}^{\Phi} \lambda m \log_2(\Phi - \phi)\mathbb{P}\left\{\Delta\Phi(t){=}\phi|\Phi(t){=}\Phi\right\} \\ &= m{-}\alpha(t){+}\lambda m\mathbb{E}\left[\log_2(\Phi{-}\Delta\Phi(t))|\Phi(t){=}\Phi\right] - \lambda m \log_2(\Phi)p_0 \quad (2)\end{aligned}$$

where we used the fact that $\sum_{\phi=1}^{\Phi}(\ldots) = \sum_{\phi=0}^{\Phi}(\ldots) - \lambda m \log_2(\Phi)p_0$.

Moreover, by Jensen's inequality (log is concave), we have:

$$\begin{aligned}\mathbb{E}\left[\log_2(\Phi - \Delta\Phi(t))|\Phi(t) = \Phi\right] &\leq \log_2(\Phi - \mathbb{E}\left[\Delta\Phi(t)|\Phi(t) = \Phi\right]) \\ &\leq \log_2(\Phi - h(\alpha(t))\Phi) \qquad (3)\end{aligned}$$

Combining equations (2) and (3), we get:

$$(1-p_0)\mathbb{E}\left[R(t)|\Phi(t) = \Phi\right] \leq (1-p_0)\lambda m \log_2(\Phi) + m - \alpha(t) + \lambda m \log_2(1 - h(\alpha(t))).$$

If $\alpha(t) = m$, the sum of the two last terms of the equation is negative since $1 - h(\alpha) \leq 1$. If $\alpha(t) = 0$, the schedule is finished. If $0 < \alpha(t) < m$, the sum of the two last terms is negative by definition of λ (λ corresponds to the worst choice of $\alpha(t)$). Dividing on both sides by $1 - p_0$ concludes the proof of *(i)*.

The proof of *(ii)* is quite similar to the proof of *(i)*. We prove by induction on Φ that $\mathbb{E}\left[2^{R(t)/(\lambda m) - \log_2(\Phi)}|\Phi(t) = \Phi\right] \leq 1$. It clearly holds for $\Phi = 1$ since in that case it is equal to 0. $\mathbb{E}\left[2^{R(t)/(\lambda m) - \log_2(\Phi)}|\Phi(t) = \Phi\right]$ is equal to

$$\sum_{\phi=0}^{\Phi} \mathbb{E}\left[2^{R(t)/(\lambda m) - \log_2(\Phi)}|\Phi(t+1){=}\Phi - \phi\right] \mathbb{P}\left\{\Delta\Phi(t){=}\phi|\Phi(t){=}\Phi\right\}$$

Using $R(t{+}1) = m{-}\alpha{+}R(t)$ and introducing $\log_2(\Phi{-}\phi){-}\log_2(\Phi{-}\phi)$, this equals

$$\begin{aligned}&\sum_{\phi=1}^{\Phi} 2^{\frac{m-\alpha}{\lambda m} + \log_2(1-\frac{\phi}{\Phi})}\mathbb{E}\left[2^{\frac{R(t+1)}{\lambda m} - \log_2(\Phi-\phi)}|\Phi(t{+}1){=}\Phi{-}\phi\right] \mathbb{P}\left\{\Delta\Phi(t){=}\phi|\Phi(t){=}\Phi\right\} \\ &\qquad + 2^{\frac{m-\alpha}{\lambda m}}\mathbb{E}\left[2^{\frac{R(t+1)}{\lambda m} - \log_2(\Phi)}|\Phi(t+1){=}\Phi\right] p_0 \\ &\leq \sum_{\phi=1}^{\Phi} 2^{\frac{m-\alpha}{\lambda m} + \log_2(1-\frac{\phi}{\Phi})}\mathbb{P}\left\{\Delta\Phi(t){=}\phi|\Phi(t){=}\Phi\right\} + 2^{\frac{m-\alpha}{\lambda m}}\mathbb{E}\left[2^{\frac{R(t)}{\lambda m} - \log_2(\Phi)}|\Phi(t){=}\Phi\right] p_0\end{aligned}$$

where we used the induction hypothesis for the inequality.

Then, adding and subtracting the first term of the sum $\sum_{\phi=1}^{\Phi}$, this leads to

$$\begin{aligned}(1-2^{\frac{m-\alpha}{\lambda m}}p_0)\mathbb{E}\left[2^{\frac{R(t)}{\lambda m}-\log_2(\Phi)}|\Phi(t){=}\Phi\right] &\le 2^{\frac{m-\alpha}{\lambda m}}\mathbb{E}\left[1-\Delta\Phi/\Phi|\Phi(t){=}\Phi\right]-2^{\frac{m-\alpha}{\lambda m}}p_0\\ &\le 2^{\frac{m-\alpha}{\lambda m}+\log_2(1-h(\alpha))}-2^{\frac{m-\alpha}{\lambda m}}p_0\\ &\le 1-2^{\frac{m-\alpha}{\lambda m}}p_0.\end{aligned}$$

where we used the definition of λ to show that the first term is less than one. This shows that $\mathbb{E}\left[2^{\frac{R(t)}{\lambda m}-\log_2(\Phi)}|\Phi(t){=}\Phi\right]\le 1$. Therefore by Markov's inequality:

$$\mathbb{P}\left\{R(t)\ge \lambda m\log_2\Phi+u|\Phi(t){=}\Phi\right\}=\mathbb{P}\left\{2^{\frac{R(t)}{\lambda m}-\log_2\Phi}\ge 2^{\frac{u}{\lambda m}}|\Phi(t){=}\Phi\right\}\le 2^{-\frac{u}{\lambda m}}.$$

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.

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 $w_i(t)=w(t)/m$, then $\Phi(t)=0$. $\Phi(t)\le 1$ 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 $\Phi(t)\le w(t)^2-w(t)^2/m\le(1-1/m)w^2(t)$.

Assume that at time t, the queue i has $w_i(t)\ge 1$ tasks. If 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, $w_i(t+1)=\lceil(w_i(t)-1)/2\rceil$ and $w_j(t+1)=\lfloor(w_i(t)-1)/2\rfloor$. Thus $w_i(t+1)^2+w_j(t+1)^2=\lceil(w_i(t)-1)/2\rceil^2+\lfloor(w_i(t)-1)/2\rfloor^2\le w_i(t)^2/2-w_i(t)+1$. Therefore, this generates a difference of potential of

$$\delta_i^k(t)=\delta_i^1(t)\ge w_i(t)^2/2+w_i(t)-1. \qquad (4)$$

If i receives zero steal requests, it potential goes from $w_i(t)^2$ to $(w_i(t)-1)^2$, generating a potential decrease of $2w_i(t)-1$. The last event contributing to the change of the potential is that $(\sum_{i=1}^m w_i(t))^2/m$ goes from $w(t)^2/m$ to $w(t+1)^2 = (w(t)-\alpha(t))^2/m$, generating a potential increase of $2\alpha(t)w(t)/m - \alpha(t)^2/m$.

Recall that at time t, there are $\alpha(t)$ active processors and therefore $m-\alpha(t)$ processors that send steal requests. A processor i receives zero steal requests if the $m-\alpha(t)$ 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 $p_r(\alpha(t))$:

$$p_r(\alpha(t)) = 1 - \left(1 - \frac{1}{m-1}\right)^{m-\alpha(t)}.$$

If $\Phi(t)=\Phi$ and $\alpha(t)=\alpha$, by summing the expected decrease on each active processor δ_i^1, the expected potential decrease is greater than:

$$\sum_{i/w_i(t)>0}\left[p_r(\alpha)\Big(\frac{w_i(t)^2}{2}+w_i(t)-1\Big)+(1-p_r(\alpha))(2w_i(t)-1)\right]-2w(t)\frac{\alpha}{m}+\frac{\alpha^2}{m}$$

$$= \frac{p_r(\alpha)}{2}\Phi + \frac{p_r(\alpha)}{2}\left(\frac{w(t)^2}{m} - 2w(t) + 2\frac{m-\alpha}{mp_r(\alpha)}\left(2w(t)-\alpha\right)\right) \qquad (5)$$

$$\geq \frac{p_r(\alpha)}{2}\left(\Phi + \frac{w(t)^2}{m} - 2\frac{w(t)}{m} + 2\left(1-\frac{1}{m}\right)(w(t)-\alpha)\right) \geq \frac{p_r(\alpha)}{2}\Phi$$

The details of the computation of (5) can be found in Appendix A.

Using Theorem 1 of the previous section, we conclude the analysis by the following theorem.

Theorem 2. *Let $C_{\max}$ be the makespan of W unit independent tasks scheduled by work stealing. Then:*

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

(ii) $$\mathbb{P}\left\{C_{\max} \geq \frac{W}{m} + \frac{2}{1-\log_2(1+\frac{1}{e})}\cdot\left(\log_2 W + \log_2\frac{1}{\epsilon}\right)+1\right\} \leq \epsilon$$

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

Proof. Equation (5) shows that $\mathbb{E}\left[\Delta\Phi(t)|\Phi(t)=\phi,\alpha(t)=\alpha\right] \leq h(\alpha)\Phi$ with $h(\alpha) = p_r(\alpha)/2$. Using Theorem 1 *(i)* and the fact that $\Phi(0)\leq W^2$, the expected number of steal requests before $\Phi(t)\leq 1$ is bounded by:

$$\mathbb{E}\left[R\right] \leq \lambda m \log_2(W^2) = 2\lambda m\log_2(W),$$

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

As said before, when $\Phi(t)\leq 1$, there is at most one processor with at least one more task than the others. Therefore, there will be a steal request only when this processor will have one task and the others zero. This happens only once and generates at most $m-1$ steal requests.

At each time step of the schedule, a processor is either computing one task or stealing work. This shows that $m \cdot C_{\max} = W + R$. Thus:

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

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, $C_{\max} \geq \frac{W}{m} + \log_2 W - 1$.

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 7 seem to indicate that the factor of $\log_2(W)$ is slightly less. This shows that the constants obtained by our analysis are sharp.

Initial repartition of tasks. In the worst case, all tasks are in the same queue at the beginning of the execution. Using bounds in terms of Φ_0, one can show that a more balanced initial repartition leads to fewer steal requests on average. Suppose that we take a balls-and-bins assignment as the initial repartition: for each task, we choose a processor at random and put the task in its queue. The expected value of Φ_0 is:

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

as w_i follows a binomial distribution. Since the number of work requests is proportional to $\log_2 \Phi_0$, this initial distribution of tasks reduces the number of steal requests by a factor of 2 on average.

5 Tasks with Precedences

In this section, we show how the well known non-blocking work stealing of Arora *et al.* [7] (denoted ABP in the sequel) can be analyzed with our method which provides tighter bounds for the makespan. Following [7], a multithreaded computation is modeled as a directed acyclic graph G with a single root node; each node corresponds to a unit task and edges define precedence constraints. The out-degree of each node is either 0, 1 or 2. The critical path of G is denoted by T_∞ and W is its total number of nodes.

ABP schedules the DAG G as follows. Each process i maintains a double-ended queue (called a deque) Q_i of ready nodes (the notion of process here corresponds to our processors). At each slot, an active process i with a non-empty deque executes the node at the bottom of its deque Q_i; once its execution is completed, this node is popped from the bottom of the deque, enabling – *i.e.* making ready – 0, 1 or 2 child nodes that are pushed at the bottom of Q_i. At each top, an idle process 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 nodes, then its top-most node 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 node currently in execution on i; or, due to contention, another thief performs a successful steal request on i simultaneously.

Let us first recall the definition of the *enabling tree* of [7]. If the execution of node u enables node 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. If $d(u)$ is the depth of a node u in the enabling tree, then its weight is defined as $\omega(u) = T_\infty - d(u)$. The weight of the root node is T_∞.

To represent the amount of work on each processor, we define $w_i(t) = 2^{\max\{\omega(u):u\in Q_i(t)\}}$, *i.e.* the maximum number of tasks that can be activated by a task in Q_i. We first study the repartition of the work during a steal request.

Lemma 1. *For any active process i, we have $w_i(t+1) \leq w_i(t)$. Moreover, after any steal request from a process j on i, $w_i(t+1) \leq \dfrac{w_i(t)}{2}$ and $w_j(t+1) \leq \dfrac{w_i(t)}{2}$.*

Proof. The proof is derived from [7], Corollary 4 in Section 3: if at t, Q_i contains the $k+1$ nodes $v_0, v_1, \ldots, v_k$ from bottom to top, then $\omega(v_0) \leq \omega(v_1) < \ldots < \omega(v_{k-1}) < \omega(v_k)$. After the execution of a node u, the maximum weight of its two enabled children is less than $\omega(u) - 1$. Thus the potential work w_i cannot increase.

We now state that the potential is halved after any steal request by distinguishing two cases. First, when a successful steal occurs on i from j, then the node v_k has been stolen and executed by j. Thus, either $w_i(t+1) = 0$ if Q_i is empty at $t+1$; or $w_i(t+1) = 2^{\omega(v_{k-1})} \leq 2^{\omega(v_k)-1} \leq w_i(t)/2$. Besides, after execution of v_k by j, $w_j(t+1) \leq 2^{\omega(v_k)-1} \leq w_i(t)/2$. Secondly, if all steal requests that occur on i are unsuccessful, then there was only one node v_0 in Q_i whose execution was processed by i. Then, at $t+1$, $w_i(t+1) \leq 2^{\omega(v_0)-1} \leq w_i(t)/2$ and $w_j(t+1) = 0$.

We can now state the following theorem.

Theorem 3. *The expected makespan of ABP work stealing verifies:*

(i) $$\mathbb{E}\left[C_{\max}\right] \leq \frac{W}{m} + \frac{2}{1-\log_2(1+1/e)} \cdot T_\infty + 1 < \frac{W}{m} + 3.65 \cdot T_\infty + 1.$$

(ii) $$\mathbb{P}\left\{C_{\max} \geq \frac{W}{m} + \frac{2}{1-\log_2(1+1/e)} \cdot \left(T_\infty + \log_2\frac{1}{\epsilon}\right) + 1\right\} \leq \epsilon$$

Proof. The proof is a direct application of Theorem 1 using the potential function $\Phi(t) = \sum_i w_i(t)^2$. Note that we cannot use the same potential as in Section 4 because the total amount of work may be reduced when a steal occurs[3].

[3] This can happen when the sibling of the stolen task has only one child.

Remark. In [7], the authors established the upper bounds

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

in Section 4.3, proof of Theorem 9. Our bounds greatly improve the constant factors of this previous result and are close to simulation values (cf. Section 7).

6 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 [5]. 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 use the potential function[4] $\Phi(t) = \sum_{i=1}^m w_i(t)^2$. 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 $w_i(t) = (k+1)q + r + 1$ with $0 \leq r < k+1$. After one time step and k steal requests, the work will be divided in r parts with $q+1$ tasks and $k+1-r$ parts with q tasks. By a direct computation, the potential generated by these steal requests at time $t+1$ can be bounded by:

$$r(q+1)^2 + (k+1-r)q^2 = (k+1)q^2 + r(2q+1) \leq \frac{1}{k+1}\left((k+1)q + r\right)^2 \leq \frac{w_i(t)^2}{k+1}.$$

If $m-\alpha$ processors send steal requests, the probability for an active processor to receive k steal requests is

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

The expected diminution of the potential caused by the steals on processor i is equal to $\sum_{k=0}^{m-\alpha} \delta_i^k p_k(\alpha)$. By a direct computation, this is bounded by

$$\begin{aligned}\sum_{k=0}^{m-\alpha} \delta_i^k p_k(\alpha) &\geq \sum_{k=0}^{m-\alpha} \left(1 - \frac{1}{k+1}\right) w_i(t)^2 p_k(\alpha) \\ &= w_i(t)^2 \left(1 - \frac{m-1}{m-\alpha+1}\left(1 - \left(\frac{m-2}{m-1}\right)^{m-\alpha+1}\right)\right)\end{aligned}$$

This shows that $\mathbb{E}\left[\Delta\Phi(t) | \Phi(t) = \Phi | \alpha(t) = \alpha\right] \leq h(\alpha)\Phi$ where

$$h(\alpha) = 1 - \frac{m-1}{m-\alpha+1}\left(1 - \left(\frac{m-2}{m-1}\right)^{m-\alpha+1}\right)$$

[4] The same potential function as in Section 4 could be used but leads to more complex computations.

Deriving with respect to α shows that $(m-\alpha)/-\log_2(1-h(\alpha))$ is decreasing. Thus $\lambda = \max_{1\leq\alpha\leq m}(m-\alpha)/-m\log_2(1-h(\alpha)) = (m-1)/-m\log_2(1-h(1))$. A direct computation shows that $\lambda \leq 1/-\log_2(1-1/e)$. See Appendix C for details. Therefore we can copy *mutatis mutandis* the proof of Theorem 2 to show that:

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

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

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

Compared to the situation with no cooperation among thieves, the number of steal requests is reduced by a factor $\frac{1-\log_2(1+1/e)}{-\log_2(1-1/e)} \approx 1.20$. We will see in Section 7 that this is close to the value obtained by simulation.

7 Experimental Study

Theorem 1 provides upper bound on the expected value of the makespan for the models considered in Sections 4,5,6. In this section, we experimentally study the constant factor of the $\log_2 W$ term and show that it is close to the theoretical result. We focus on independent tasks as it is difficult to generate a realistic random DAG. Moreover, the DAG with the maximum number of tasks, out-degree at most 2 and critical path T_∞ is a complete binary tree of height T_∞. This is the worst case for the bound given in Section 5 and it is similar to the independent tasks case.

We developed a simulator that strictly follows the model of Sections 4 and 6. 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%.

We computed the constant factor 2λ of the $\log_2 W$ term for various number of processors and tasks. The value goes to a limit between 2 and 3 (cf. Fig. 1). This gives a constant $2\lambda \approx 2.37$ for unit independent tasks with standard steal and $2\lambda_{\text{coop}} \approx 2.08$ for unit independent tasks with cooperative steal. The theoretical values of 3.65 (standard steal) and 3.02 (cooperative steal) are close, only 50% greater than the simulation values. The difference can be explained by the use of an adversary in Theorem 1. Moreover, the analysis is fine enough to predict the advantage of the cooperative steal with a gain of 20% over the standard steal, close to the experimental gain of 14%.

8 Concluding Remarks

We have presented in this paper a new analysis of work stealing. The main result is to prove that the expected makespan to schedule a workload of W on m processors is no more than the best possible absolute lower bound W/m plus an additive term in $3.65\log_2 W$ very close to the value obtained by simulation.

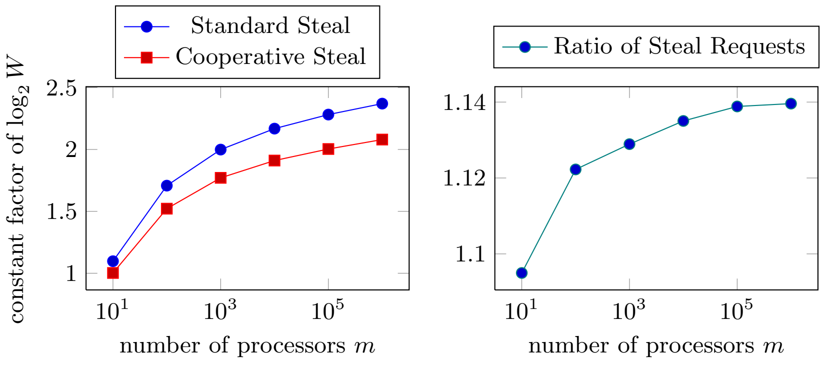

Fig. 1. (Left) Constant factor of $\log_2 W$ against the number of processors for the standard steal and the cooperative steal. (Right) Ratio of steal requests (standard/cooperative).

References

1. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics **17** (1969) 416–429
2. Hoffmann, R., Korch, M., Rauber, T.: Performance evaluation of task pools based on hardware synchronization. In: Proc. of Supercomputing. (2004)
3. Frigo, M., Leiserson, C.E., Randall, K.H.: The implementation of the Cilk-5 multithreaded language. In: Proceedings of PLDI. (1998)
4. Robison, A., Voss, M., Kukanov, A.: Optimization via reflection on work stealing in TBB. In: Proceedings of IPDPS. (2008) 1–8
5. Gautier, T., Besseron, X., Pigeon, L.: KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In: Proceedings of PASCO. (2007) 15–23
6. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. Journal of the ACM **46**(5) (1999) 720–748
7. Arora, N.S., Blumofe, R.D., Plaxton, C.G.: Thread scheduling for multiprogrammed multiprocessors. Theory of Computing Systems **34**(2) (2001) 115–144
8. Berenbrink, P., Friedetzky, T., Goldberg, L.A.: The natural work-stealing algorithm is stable. SIAM Journal of Computing **32**(5) (2003) 1260–1279
9. Mitzenmacher, M.: Analyses of load stealing models based on differential equations. In: Proceedings of SPAA. (1998) 212–221
10. Hendler, D., Shavit, N.: Non-blocking steal-half work queues. In: Proceedings of PODC. (2002)
11. Gast, N., Gaujal, B.: A Mean Field Model of Work Stealing in Large-Scale Systems. In: Proceedings of SIGMETRICS. (2010)
12. Berenbrink, P., Friedetzky, T., Goldberg, L.A., Goldberg, P.W., Hu, Z., Martin, R.: Distributed selfish load balancing. SIAM Journal on Computing **37**(4) (2007)
13. Traoré, D., Roch, J.L., Maillard, N., Gautier, T., Bernard, J.: Deque-free work-optimal parallel STL algorithms. In: Proceedings of Euro-Par. (2008) 887–897
14. Puterman, M.L.: Markov Decision Processes : Discrete Stochastic Dynamic Programming. Wiley (2005)

A Proof of Inequality 5 of Theorem 2

In this section, we show that in the case of independent tasks, the expectation of the potential decrease is greater than $\Phi p_r(\alpha)/2$.

Recall that the expectation of the potential decrease is greater than:

$$\sum_{i/w_i(t)>0}\left[p_r(\alpha)\left(\frac{w_i(t)^2}{2}+w_i(t){-}1\right)+(1{-}p_r(\alpha))(2w_i(t){-}1)\right]-2w(t)\frac{\alpha}{m}+\frac{\alpha^2}{m}$$
$$=\frac{p_r(\alpha)}{2}\left(\sum w_i(t)^2-\frac{w(t)^2}{m}\right)+p_r(\alpha)\left(\frac{w(t)^2}{2m}+w(t)-\alpha\right)$$
$$+(1-p_r(\alpha))\left(2w(t)-\alpha\right)-2w(t)\frac{\alpha}{m}+\frac{\alpha^2}{m}$$

where we used the fact that $\sum_{i/w_i(t)>0}w_i(t)=w(t)$ and that $\sum_{i/w_i(t)>0}1=\alpha$ since α is the number of active processors. A direct computation shows that this is equal to

$$\frac{p_r(\alpha)}{2}\Phi+p_r(\alpha)\left(\frac{w(t)^2}{2m}+w(t)-\alpha-2w(t)+\alpha\right)+2w(t)-\alpha-2w(t)\frac{\alpha}{m}+\frac{\alpha^2}{m}$$
$$=\frac{p_r(\alpha)}{2}\Phi+\frac{p_r(\alpha)}{2}\left(\frac{w(t)^2}{m}-2w(t)+\frac{2}{p_r(\alpha)}\left(1-\frac{\alpha}{m}\right)(2w(t)-\alpha)\right)$$
$$=\frac{p_r(\alpha)}{2}\Phi+\frac{p_r(\alpha)}{2}\left(\frac{w(t)^2}{m}-2w(t)+\frac{2}{m}\frac{m-\alpha}{p_r(\alpha)}(2w(t)-\alpha)\right)\tag{6}$$

Let define f by

$$f(\alpha)\overset{\text{def}}{=}\frac{m-\alpha}{p_r(\alpha)}=\frac{m-\alpha}{1-\left(1-\frac{1}{m-1}\right)^{m-\alpha}}$$

and compute the derivative f'.

$$\left(1-\left(1-\frac{1}{m-1}\right)^{m-\alpha}\right)^2\cdot f'(\alpha)=-\left(1-\left(1-\frac{1}{m-1}\right)^{m-\alpha}\right)$$
$$+(m-\alpha)\cdot\ln\left(1-\frac{1}{m-1}\right)\cdot\left(1-\frac{1}{m-1}\right)^{m-\alpha}$$
$$=-1+\left(1-\frac{1}{m-1}\right)^{m-\alpha}\cdot\left(1+(m-\alpha)\ln\left(1-\frac{1}{m-1}\right)\right)$$
$$\le-1+\left(1-\frac{1}{m-1}\right)^{m-\alpha}\cdot\left(1+(m-\alpha)\frac{-1}{m-1}\right)$$
$$\le-1+\left(1-\frac{1}{m-1}\right)^{m-\alpha}\cdot\frac{\alpha-1}{m-1}\le 0$$

As f is non increasing for $1\le\alpha\le m-1$,

$$\min_{1\le\alpha\le m-1}f(\alpha)=f(m-1)=\frac{1}{1-\left(1-\frac{1}{m-1}\right)^1}=m-1.$$

(6) is greater than:

$$\frac{p_r(\alpha)}{2}\Phi+\frac{p_r(\alpha)}{2}\left(\frac{w(t)^2}{m}-2w(t)+2\frac{m-1}{m}(2w(t)-\alpha)\right)$$
$$=\frac{p_r(\alpha)}{2}\Phi+\frac{p_r(\alpha)}{2}\left(\frac{w(t)^2}{m}-2w(t)+2w(t)\left(1-\frac{1}{m}\right)+2\left(1-\frac{1}{m}\right)(w(t)-\alpha)\right)$$
$$=\frac{p_r(\alpha)}{2}\Phi+\frac{p_r(\alpha)}{2}\left(\frac{w(t)^2}{m}-\frac{2w(t)}{m}+2\left(1-\frac{1}{m}\right)(w(t)-\alpha)\right)$$

As $w(t) - \alpha(t) \geq 0$ (an active processor has at least one task) the last term is positive. Moreover, for all $w(t) > 1$, the second term is positive. Thus, this is greater than $\frac{p_r(\alpha)}{2}\Phi$ which concludes the proof of the inequality:

$$\sum_{i/w_i(t)>0} \left[p_r(\alpha) \left(\frac{w_i(t)^2}{2} + w_i(t) - 1 \right) + (1 - p_r(\alpha))(2w_i(t) - 1) \right] - 2w(t)\frac{\alpha}{m} + \frac{\alpha^2}{m} \geq \frac{p_r(\alpha)}{2}\Phi$$

B Computation of λ for the unit tasks

In this section, we compute the constant λ for the unit tasks. We first show that the quantity $(m - \alpha)/(-\log_2(1 - p_r(\alpha)/2))$ is decreasing in α. Then we bound the value $(m - 1)/(-\log_2(1 - p_r(1)/2))$ by $(m - 1)/(1 - \log_2(1 + 1/e))$.

Let $g(\alpha) \overset{\text{def}}{=} -\log_2(1 - p_r(\alpha)/2)$ and $f(\alpha) \overset{\text{def}}{=} (m - \alpha)/g(\alpha)$. By definition of $p_r(\alpha)$, $g(\alpha)$ can be written:

$$g(\alpha) = -\log_2 \left(\frac{1}{2} + \frac{1}{2} \left(1 - \frac{1}{m-1} \right)^{m-\alpha} \right) = 1 - \log_2 \left(1 + \left(1 - \frac{1}{m-1} \right)^{m-\alpha} \right).$$

Denoting $p \overset{\text{def}}{=} 1 - 1/(m - 1)$ and $x \overset{\text{def}}{=} p^{m-\alpha}$, the derivative of f with respect to α is:

$$\begin{aligned} f'(\alpha) &= \frac{\ln(1 + p^{m-\alpha}) - \ln 2 + p^{m-\alpha} \left(\ln(1 + p^{m-\alpha}) - \ln 2 \right) + p^{m-\alpha} \ln(p)(\alpha - m)}{(1 + p^{m-\alpha})g(\alpha)^2 \ln 2} \\ &= \frac{(\ln(1 + x) - \ln 2)\, x + \ln(1 + x) - x \ln x - \ln 2}{(1 + x)g(\alpha)^2 \ln 2} \\ &= \frac{(1 + x) \ln(1 + x) - x \ln x - (1 + x) \ln 2}{(1 + x)g(\alpha)^2 \ln 2} \end{aligned}$$

The derivative of $(1+x)\ln(1+x) - x \ln x - (1+x) \ln 2$ w.r.t. x is $\ln(1+x) - \ln(x) - \ln 2 = \ln(1+1/x) - \ln 2 > 0$. As $x < 1$, this shows that $(1+x)\ln(1+x) - x \ln x - (1+x) \ln 2 < (1 + 1)\ln(1 + 1) - 1 \ln 1 - (1 + 1) \ln 2 = 0$.

Thus, $f(\alpha)$ is decreasing and

$$\lambda = \max_{1 \leq \alpha \leq m-1} \frac{1}{m} f(\alpha) = \frac{1}{m} f(1) \leq \frac{1}{1 - \log_2 \left(1 + \left(1 - \frac{1}{m-1} \right)^{m-1} \right)}$$

Using the fact that for all $m \geq 2$:

$$\left(1 - \frac{1}{m-1} \right)^{m-1} = \exp \left((m - 1) \ln(1 - \frac{1}{m-1}) \right) \leq \exp \left(-(m - 1)\frac{1}{m-1} \right) = \frac{1}{e},$$

we get that $1 - \log_2 \left(1 + (1 - 1/(m - 1))^{m-1} \right) \geq 1 - \log_2(1 + 1/e)$. This shows that

$$\lambda \leq \frac{1}{1 - \log_2(1 + 1/e)}$$

C Computation of λ for the cooperative steal

In this section, we compute the maximum of $(m-\alpha)/-\log_2(1-h(\alpha))$ for $1 \le \alpha \le m-1$ in the case of cooperative thieves. We first show this function is decreasing in α and then we bound its value in for $\alpha = 1$.

Let $g(\alpha) \overset{\text{def}}{=} -\log_2(1-h(\alpha))$ with $h(\alpha)$ defined as in Section 6. $g(\alpha)$ is equal to

$$g(\alpha) = -\log_2\left(\frac{m-1}{m-\alpha+1}\left(1-\left(1-\frac{1}{m-1}\right)^{m-\alpha+1}\right)\right)$$

Let $f(\alpha) \overset{\text{def}}{=} (m-\alpha)/g(\alpha)$. Let $f'(\alpha)$ be the derivative of $f(\alpha)$ w.r.t. α. Denoting $p \overset{\text{def}}{=} 1 - 1/(m-1)$ and $n \overset{\text{def}}{=} m-\alpha+1$, we define $k(p)$ by:

$$k(p) \overset{\text{def}}{=} f'(\alpha)g^2(\alpha)\ln 2 = -\ln(n(1-p)) + \ln(1+p^n) + \frac{n-1}{n}\left(1+\frac{p^n\ln(p)n}{1-p^n}\right)$$

The derivative of $k(p)$ w.r.t. p is

$$\begin{aligned} k'(p) &= \frac{\ln(p^n)(n-1)p^{n-1}(p-1)+(p^n-1)(1-p^{n-1})}{(p-1)(p^n-1)^2} \\ &= \frac{1-p^{n-1}}{(p-1)(p^n-1)^2}\left(p^{n-1}\ln(p^n)\frac{(1-n)(p-1)}{1-p^{n-1}}+p^n-1\right) \end{aligned}$$

Moreover, $(1-p^{n-1})/(1-p) = 1+p+p^2\cdots+p^{n-2} \le n-1$ and since $\ln(p^n) < 0$ and $p^{n-1} > p^n$, we have:

$$\ln(p^n)p^{n-1}\frac{(1-n)(p-1)}{1-p^{n-1}} + p^n - 1 \le p^{n-1}\ln(p^n) + p^n - 1 \le p^n\ln(p^n) + p^n - 1.$$

Since $p^n < 1$, $p^n\ln(p^n) < 0$ and $p^n - 1 < 0$. Thus, the last part of the equation is negative. Since $1-p$ is negative, $k'(p) \ge 0$. This shows that $k'(p) \le k(1) = 0$. Therefore $f'(\alpha) \le 0$ and $f(\alpha) \ge f(1) = (m-1)/g(1)$ for $1 \le \alpha \le m-1$. We have:

$$\begin{aligned} g(1) &= -\log_2\frac{m-1}{m} - \log_2\left(1-\left(1-\left(\frac{1}{m-1}\right)\right)^m\right) \\ &\ge -\log_2(1-e^{-1}). \end{aligned}$$

This shows that

$$\lambda = \frac{1}{m}f(1) \le \frac{1}{m}\frac{m-1}{-\log_2(1-\frac{1}{e})} \le \frac{1}{-\log_2(1-\frac{1}{e})}.$$