A robust randomized algorithm to perform independent tasks

Bogdan S. Chlebus, Leszek Gąsieniec, Dariusz R. Kowalski, Alex A. Shvartsman

A Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO 80217, USA
B Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK
C Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA
D Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

A R T I C L E   I N F O

Article history:
Received 5 June 2003
Accepted 12 March 2008
Available online 29 March 2008

Keywords:
Randomization
Distributed algorithm
Message passing
Crash failure
Scheduling tasks
Ramanujan graphs

A B S T R A C T

The Do-All problem is about scheduling $t$ similar and independent tasks to be performed by $p$ processors prone to crashes. We assume that the distributed system is synchronous with processors communicating by message passing. Crashes are determined by a fully adaptive adversary that is restricted only by an upper bound $f$ on the number of crashes. The complexity of algorithms is measured by work and communication, where work is defined as the number of available-processor steps, and communication as the number of point-to-point messages. We develop a randomized algorithm with $W = O(t + p \cdot \log^2 p \cdot \log \log p)$ expected work and $O((p - f)^3 \log W)$ expected communication, for an arbitrary number $f < p$ of crashes.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Scheduling tasks to be performed by failure-prone processors of a distributed system is among classical problems of fault-tolerant computing. The specific problem we consider is called Do-All: tasks are similar, independent and idempotent. Properties and inter-dependencies of tasks do not matter in solution of Do-All, but rather combinatorial aspects of the problem and properties of the underlying distributed system.

We assume that the distributed system is synchronous and that processors communicate by sending messages. Crash failures are imposed by an adversary that is assumed to be adaptive, in the sense that decisions about which specific processors to fail are made on-line in the course of an execution. An adaptive adversary can make a decision at a round in an execution that depends on the random bits generated by this round. The behavior of adversary is subject only to an upper bound $f$ on the number of crashes; this adversary is called $f$-bounded. We assume that in any execution at least one processor stays operational without failing. The most powerful adversary we consider, for which this is the only restriction, is called unbounded.


* Corresponding author.
E-mail address: bogdan.chlebus@odu.edu (B.S. Chlebus).
1 The work of this author was supported by NSF Grant 0310503.
2 A part of this work had been done while the author was a post-doctoral researcher at the University of Connecticut and was supported by NSF-NATO Grant 0209588.
3 The work of this author was supported by NSF Career Award and NSF Grants 9988304, 0121277, 0311368, and 0702670.
Table 1
Summary of known solutions of Do-All in the message-passing model. Crashes are not assumed to be clean, unless stated otherwise. The number \( f < p \) of failures is not a part of code, unless stated otherwise. Algorithms are deterministic and explicit, unless stated otherwise.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Metrics</th>
<th>Complexity</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7]</td>
<td>Work (tasks)</td>
<td>( O(t + p) )</td>
<td>Task-oriented work (idling and waiting not counted)</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>( O(p^3) )</td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>Work</td>
<td>( O(t + p^2) )</td>
<td></td>
</tr>
<tr>
<td>[9]</td>
<td>Work</td>
<td>( O(f^{p^e} + p \log p) )</td>
<td>Numbers ( f ) and ( e ) are part of code</td>
</tr>
<tr>
<td>[1]</td>
<td>Work</td>
<td>( O(t \log f + p \log \log p) )</td>
<td>Clean crashes; there are no restarts</td>
</tr>
<tr>
<td>[1]</td>
<td>Messages</td>
<td>( O(t + p \log f + f \log p) )</td>
<td>Clean crashes; restarted for less than ( p ) restarts</td>
</tr>
<tr>
<td>[2]</td>
<td>Work</td>
<td>( W = O(t + p \sqrt{\frac{t}{p} \log^{\frac{3}{2}} p}) )</td>
<td>Number ( f ) a part of code</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>( O(\sqrt{W \cdot \frac{p}{t}}) )</td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>Work</td>
<td>( O(t + p^{1.77}) )</td>
<td>Existential solution, optimized for effort</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>( O(t + p^{1.77}) )</td>
<td></td>
</tr>
<tr>
<td>[10]</td>
<td>Work</td>
<td>( O(t + p^{1+\epsilon}), \text{any } \epsilon &gt; 0 )</td>
<td>Existential, optimized for effort, ( f ) a part of code</td>
</tr>
<tr>
<td>[2]</td>
<td>Work</td>
<td>( W = O(t + p \log^2 p) )</td>
<td>Existential solution number ( f ) a part of code</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>( O(\sqrt{W \cdot \frac{p}{t}}) )</td>
<td></td>
</tr>
</tbody>
</table>

This paper

| Work     | \( W = O(t + p \log^2 p) \) | Randomized solution, number \( f \) a part of code |
| Messages | \( O(\sqrt{W \cdot \frac{p}{t}}) \) | |

We present algorithm RANDOMIZED-Do-All that uses randomized and deterministic task allocation policies during different stages of computation. Let \( p \) be the number of processors, \( f < p \) an upper bound on the number of crashes, and \( t \) the number of tasks. Our algorithm has these parameters as part of its code. The algorithm attains \( W = O(t + p \log^2 p) \) expected work and \( O(\sqrt{W \cdot \frac{p}{t}}) \) communication.

Previous work. The Do-All problem was first studied by Dwork, Halpern and Waarts [7]. The complexity of solutions of the Do-All problem can be measured by various metrics, among which work and communication have been most popular. While work can be defined in various ways, communication typically denotes the number of point-to-point messages exchanged by the processors. The most successful algorithmic paradigm used in prior research was to disseminate local knowledge among groups of processors by designating coordinators whose job is first to collect and next spread the information about the progress made among all the processors.

The principal measure of algorithm performance in [7] is that of task oriented work, in which performing a task by a processor contributes a unit. This paper also considers effort, defined as the sum of the task-oriented work and the communication. One of the protocols in [7] has the processors send \( O(p^{3/2}) \) messages and takes \( O(p^2 + t) \) time. Another protocol makes the processors send \( O(t + p \log p) \) messages and takes an exponential \( O(p(p + \log p)(p + t)p^{1+\epsilon}) \) time. The third algorithm guarantees \( O(p^2(f + 1)) \) messages and terminates in \( O((f + 1)t/p + f) \) time. All of these algorithms are optimal with respect to the number of tasks that are performed, including multiplications, which is \( O(p + t) \).

De Prisco, Mayer and Yung [6] were the first to use the available-processor steps as a work-type metric. They gave an algorithm with \( O(t + (f + 1)p) \) work and simultaneous \( O((f + 1)p) \) communication. Galil, Mayer and Yung [9] gave a related algorithm with the improved \( O(f p^e + \min(f + 1, \log p)p) \) communication cost, for any \( e > 0 \). Chlebus, De Prisco and Shvartsman [1] developed algorithms based on aggressive coordination, where the number of new coordinators grows exponentially following crashes of all the previous coordinators. Their algorithms rely on the property of crashes to be clean, which means that if a sender fails during a multicast, then either none or all of the messages are delivered to their recipients. One of the algorithms given in [1] has \( O((t + p \log p/\log \log p) \log f) \) work and \( O(t + p \log p/\log \log p + f) \) communication. These bounds were improved in [11] for \( f \leq p/\log \log p \). Another algorithm in [1] incorporates restarted processors; it is the only known algorithm able to deal with restarts efficiently, and it uses \( O((t + p \log p + f) \cdot \min(\log p, \log f)) \) work and its message complexity is \( O((t + p \log p + f)) \).

Effort defined as the task oriented work plus the communication was first used in [7]. Chlebus, Gaśieniec, Kowalski, and Shvartsman [2] considered the available-processor steps and communication as comparable resources and advocated effort, defined to mean the sum (the available-processor steps + communication), as a joint complexity measure. They presented an algorithm of \( O(t + p^{1.77}) \) effort against the unbounded adversary. This was the first algorithm achieving effort efficiency subquadratic in \( p \) for the unbounded adversary; before that work such performance has not been achieved even for a linearly-bounded adversary who crashes up to a constant fraction of the processors. The result was improved by Georgiou, Kowalski and Shvartsman [10] who developed an algorithm with \( O(t + p^{1+\epsilon}) \) work, for any fixed constant \( \epsilon \) that is a part of code of the algorithm, by an approach based on gossiping. Paper [2] presented an algorithm with \( O(t + p^{2+\epsilon}) \) work against the \( f \)-bounded adversary, for any \( f < p \), where \( f \) is a part of code of the algorithm. This result is close to a known
lower bound $\Omega(t + p \log p/\log\log p)$, see [3,11]. The algorithm achieves $O(t + p \log^2 p)$ effort against linearly-bounded adversaries.

A summary of previous results most closely related to this work is given in Table 1. This collection of results does not include research dealing with specialized adversaries and models of communication. Next we give an overview of such work. Chlebus and Kowalski [3] studied Do-All with crash failures in the presence of weakly-adaptive linearly-bounded adversaries. (An $f$-bounded adversary is called weakly-adaptive when it selects $f$ failure-prone processors prior to the start of an execution and next may crash only these selected processors at any time in the course of an execution.) They developed a randomized algorithm with the expected $O(t + p \cdot (1 + \log^p p - \log^p (p/t)))$ effort. This expected complexity is provably better than the worst-case performance of any deterministic algorithm in such a setting, because of the lower bound $\Omega(p \log p/\log\log p)$ on the worst-case work given in [3,11]. Additionally, Chlebus and Kowalski showed in [3] that a deterministic algorithm that schedules tasks by balancing them perfectly and relying on full communication to guarantee this, performs work $O(p \log p/\log\log p)$ if pitched against a linearly-bounded adversary. This implies that $\Theta(p \log p/\log\log p)$ is precisely the optimum amount of work in such a setting. Kowalski and Shvartsman [13] studied the Do-All problem in the model of asynchronous message-passing, when every message delay is at most $d$, for some parameter $d$. They showed the lower bound $\Omega(t + pd\log^d p)$ on the expected work. They developed a deterministic algorithm of work $O((t + pd) \log p)$. Georgiou, Russell and Shvartsman [12] studied an on-line version of Do-All, which they called Omni-Do, in the asynchronous system with partitionable networks. They presented a randomized algorithm achieving a competitive ratio $1 + cw/e$ against the oblivious adversary, where the computational width $cw$ is a certain parameter of the poset according to which the adversary splits and merges groups. The same authors [11] considered an iterative version of Do-All modeling a repeated use of Do-All solutions. Fernández, Georgiou, Russell, and Shvartsman [8] considered Do-All in the model of Byzantine failures. Chlebus, Kowalski and Lingas [4] and Clementi, Monti and Silvestri [5] investigated the Do-All problem in distributed systems in which communication is by way of a multiple-access channel. Paper [4] considered the impact of collision detection and randomization on the work complexity of protocols. The solutions to Do-All in [4] were required to be correct against the unbounded adversary. Paper [5] studied $F$-reliable protocols that are correct if the number of crashes is at most $F$, for a parameter $F < p$. They obtained tight bounds on the time and work of $F$-reliable deterministic protocols.

Document structure. Section 2 presents the distributed-computing environment we use, including the adversarial model of failures and complexity measures. Communication by way of embedded graphs is discussed in Section 3. A randomized algorithm, called RANDOMIZED-DoALL, is described in Section 4. The complexity of algorithm RANDOMIZED-DoALL is estimated in Section 5. We conclude with a short discussion in Section 6.

There is a companion paper [2] about deterministic solutions to Do-All. That paper introduced a generic solution to Do-All, of which algorithm RANDOMIZED-DoALL given in this paper is a randomized instantiation. We rely on some technical results presented in [2], on the communication scheme represented by embedded graphs and the properties of the generic algorithm, when describing the communication graphs in Section 3, when estimating the communication complexity of the algorithm in terms of its expected work and number of failures, and finally in the analysis of the randomized algorithm given in Section 5.

2. Technical preliminaries

In this section we discuss the distributed setting and the Do-All problem.

Distributed system. We consider a synchronous distributed system of $p$ processors. Each processor has a unique name in the interval $[1,\ldots,p]$. Every processor knows the value of $p$, in that $p$ may occur as a part of code. An execution is structured as a sequence of rounds, each consisting of a fixed number of ticks of a global-clock.

Communication. Processors communicate by sending messages. We assume that any two operational processors can exchange messages. An event of sending a message from processor $x$ to processor $y$ does not indicate the existence of a physical link connecting $x$ with $y$. Instead, this is interpreted as a routing task implemented on the network level. Still we treat an event in which processor $x$ requests the communication layer to send a message from $x$ to processor $y$ as a single point-to-point message, although this may need to be implemented as a sequence of hops of the message along the physical links of the underlying network. A processor may send a message to any set of processors during a round. A message sent at a round is delivered to its recipients within the round. A processor may receive within one round all the messages delivered to it in the previous round.

Failures. Processors are prone to crashes: a process that fails stops immediately any activity in the execution. We denote by $f$ the maximum number of failures that may occur in an execution. We do not consider processor restarts (cf. [1]), and so $f$ does not exceed $p$. The failures are unrestricted (not assumed to be clean), in the sense that if a processor crashes while attempting to multicast a message, then an arbitrary subset of the set of the intended recipients may receive the message. We assume that no messages are lost nor corrupted while in transit.

A processor may voluntarily stop to work, we say that it halts, or it may be forced to do so by the adversary if it crashes. Processors may halt at different rounds. Halted processors are considered non-faulty. A processor that neither halted nor crashed is called active. An algorithm terminates at a given round if this is the first round by which every processor either halted or crashed, which is the same first round in which there is no active processor.

Adversaries. Occurrences of failures are governed by adversarial models. An adversary is called size-bounded if there is an upper bound on the number of faults; when this bound is given as an explicit parameter $f$, then the adversary is called...
We always assume that the number $f$ of failures satisfies $f < p$, and so $p - 1$ is the maximum number of failures in an execution. An adversary is called linearly-bounded if it is $f$-bounded such that $p - f = \Omega(p)$, for $f$ being a function of $p$. The $(p - 1)$-bounded adversary is called unbounded. The word ‘unbounded’ is to convey the property that with the $(p - 1)$-bounded adversary an arbitrary number of processors could actually be failed, since if the number of failures were larger than $p - 1$, that is, there were $p$ failures, then there would no problem to consider as there would be no operational processors left. An adversary is called adaptive if it decides on-line which processors to crash and at which event in the course of an execution. In particular, if the algorithm is randomized, then the adversary can make decisions based on the random bits already generated and used. All the adversaries we consider are adaptive, subject only to the restriction of being size-bounded.

Tasks. There are $t$ tasks that need to be performed by the system. We assume the following three properties of tasks.

- Similarity: it takes exactly one round to perform a task by a processor.
- Idempotence: a task may be performed many times, possibly concurrently.
- Independence: tasks may be performed in arbitrary order by any processors.

Each task has its unique name in the interval $\{1, \ldots, t\}$. Every processor can perform a task given the name of the task. Every processor knows the values of $t$, in that $t$ may occur as a part of code. We place no restrictions on the mutual magnitude of $p$ and $t$.

**Correctness.** An algorithm solves the Do-All problem against adversary $A$ or is correct against adversary $A$ if, in any execution of the algorithm in which crashes are consistent with the power of adversary $A$, the following conditions are satisfied:

1) Every task is eventually performed by some processor.
2) Every processor eventually halts, unless it crashes.

When we state just correctness or being a Do-All solution without mentioning the adversary, we mean the unbounded adversary. Observe some properties of this strongest notion of correctness. First, a Do-All solution has the property that every task is eventually performed by some processor even if only one processor remains operational in an execution. Second, when some processor $v$ participating in an execution of an algorithm solving Do-All halts, then all the tasks have already been performed. This is because otherwise the unbounded adversary could immediately fail all the remaining processors, except for $v$, and some tasks would remain outstanding forever. This may be expressed informally as a property that when a processor halts, then it ‘knows’ that all the tasks have been performed.

We extend the notion of correctness to randomized algorithms along similar lines. Randomized algorithms solving Do-All must guarantee that all the tasks are performed in all the executions against any behavior of the unbounded adversary. The benefit of randomness is to improve the average complexity without compromising correctness as it is understood in the deterministic case. In particular, when some tasks could be left outstanding, with possibly a small probability of this occurring, then such a randomized algorithm is not considered to be a solution.

**Complexity.** We consider the following measures of performance: work complexity and communication complexity. Work $W$ counts the available-processor steps, that is, the total number of rounds of all the processors, including idling, which is accrued by each processor until its crash or termination of the algorithm. In particular, halted processors are conservatively assumed to continue contributing to work after halting, until all the processors either halt or crash, but crashed processors contribute only until crashing. Communication $M$ counts the total number of point-to-point messages sent. We define effort as $W + M$.

### 3. Embedded graphs

In this section we describe graphs used by processors to limit the amount of effort spent on solving Do-All. These graphs are conceptual only and do not represent physical networks used for communication. The same communication scheme is used in [2].

We consider simple graphs: they are undirected and without multiple edges or loops. For graph $G = (V, E)$, part $V$ is its set of nodes and $E$ the set of edges. The number of nodes in graph $H$ is denoted by $|H|$. A node $x$ connected by an edge with node $y$ is a neighbor of $y$. We denote by $G^k = (V, E^k)$ the $k$th power of graph $G$: a pair of nodes $u$ and $v$ in $V$ is an edge in $E^k$ if and only if there is a path between $u$ and $v$ in $G$ of length at most $k$.

The communication scheme of the algorithm uses communication graph $G(p)$. Each processor is considered to be a node of this graph. Pairs of processors that communicate regularly with one another constitute the edges of the graph. The topology of $G(p)$ at the start of an execution depends on the number $f$ of failures. Moreover, each faulty or halted processor is removed from the communication graph, which causes the graph to evolve through a sequence of subgraphs. Processors may have different views of the current state of the communication graph in the course of an execution, depending on their local knowledge.

Before we specify communication graphs $G(p)$, we formulate their relevant properties. Graphs $G(p)$ have a uniform upper bound on the degrees of nodes. We denote by $\Delta$ the maximum node degree of each graph $G(p)$. Let the abbreviation $g(p)$ stand for the function $30 \log_2 p + 2$ of variable $p$. Following [2,16], we say that graph $G(p)$ satisfies the subgraph...
Let $f$ be an upper bound on the number of failures. We say that a subgraph $H \subseteq G(p)$ is compact if $|H| \geq (p - f)/7$ and the diameter of $H$ is not larger than $g(p)$. Notice that the notions of the subgraph property and the compact graph are parametrized by the number $f$ of failures. Suppose that $(G_i)_{i \geq 1}$ is a sequence of subgraphs of $G(p)$ such that $G_{i+1} \subseteq G_i$ and $|G_i| \geq p - f$. We may use the subgraph property to obtain a sequence $(H_i)_{i \geq 1}$ of compact subgraphs in $G(p)$ such that $H_{i+1} \subseteq H_i$. To this end it is sufficient to define $H_i = P(G_i)$. Function $P$ is not implemented in any way, and nodes of the graph do not need to know if they are in any of the subgraphs $H_i$.

Compact graphs guarantee rapid progress in terms of the number of performed tasks, because they contain many nodes that can communicate among themselves quickly. They do not limit communication by their definition which does not stipulate any restriction on the number of edges. For instance, the complete network on $p$ nodes satisfies the subgraph property, but if it were used to determine communication among the processors, then the number of messages would be large, even if the number of failures were small. What we need are sparse compact graphs.

Denote by $L(n, d)$, for positive integers $n$ and $d$, the Ramanujan graph on $n$ nodes and of node degree $d$ introduced by Lubotzky, Phillips and Sarnak [14]. Graphs $L(n, d)$ are explicit in that they can be found in time polynomial in $n$ and $d$. We use a fixed node degree $\Delta_0$, which needs to have the property that $\Delta_0 - 1$ is a prime. For any $p$ there is an integer $n_p$ such that graph $L(n_p, \Delta_0)$ exists. We may assume that $n_p = p$ to simplify notation, since otherwise each node could simulate $O(1)$ nodes of $L(n_p, \Delta_0)$, see [2]. Graphs $L(p, \Delta_0)$ for a fixed $\Delta_0$ are denoted by $L(p)$.

**Lemma 1.** (See [2].) For every $f < p$ there exists a positive integer $\ell$ such that graph $L(p)^\ell$ has the subgraph property. The maximum degree $\Delta$ of graph $L(p)^\ell$ is $O((\frac{p}{n})^2 \log_\rho \Delta_0)$, for some constant $\rho$, which can be taken as $\rho = 27/2$ for $\Delta_0 = 74$.

For given $p$ and $f$, graph $L(p)^\ell$, for the smallest number $\ell$ for which Lemma 1 holds, is taken as the communication graph $G(p)$. We use $\Delta_0 = 74$ and $\rho = 27/2$ as in [2].

### 4. Algorithm

In this section we describe a randomized algorithm **RANDOMIZED-DOALL** to solve Do-All. The algorithm has numbers $p$, $f$ and $t$ in its code. Since the numbers $p$ and $f$ determine graph $L(p)^\ell$, as referred to in Lemma 1, the communication graph $G(p)$ is also a part of code of the algorithm.

Let the range of a processor $v$ be a subgraph of graph $G(p)$ that contains each operational processor whose distance from $v$ is at most $g(p)$. A processor is said to be compact if its range is such. These notions are dynamic and depend on occurrences of failures, so the status of being compact may be lost by a processor at some point of execution.

Algorithm **RANDOMIZED-DOALL** starts with all processors initializing their local variables and next iterating a repeat loop, see Fig. 1. One iteration of the loop is called a phase. Phases are of two kinds, called main and closing, see Figs. 2 and 3. Each phase consists of three rounds: (1) receiving messages, (2) local computation, and (3) multicasting messages. The goal of the main phase to perform an outstanding task and then forward the local knowledge about active processors and outstanding tasks to the neighbors in the communication graph. The goal of the closing phase is to notify the processors that all the tasks have been completed. We say that a processor is busy at a certain round if its local knowledge does not imply that all the tasks have been done. Initially a processor is busy since it considers all the tasks as outstanding. A busy processor executes the main phase. When a processor learns that all the tasks have been performed, it stops performing tasks and switches to the closing phase.

**Local states of processors.** Processors have local variables that determine their states. We use the convention that, for any variable, say, $X$, the local copy of $X$ at processor $v$ is denoted by $X_v$.

Every processor $v$ maintains the following three ordered lists. List $\text{Tasks}_v$ contains the tasks assumed by $v$ to be still outstanding. List $\text{Processors}_v$ contains the names of processors assumed by $v$ to be active, that is, neither halted nor

---

Fig. 1. Algorithm **RANDOMIZED-DOALL**. Code for processor $v$. 

---
crashed. Finally, list \( \text{Busy}_v \) contains the names of processors assumed by \( v \) to be busy performing tasks in the main phase. These three lists are initially ordered by the names of items they store. The position of an item in a list is its rank in the list. Items may be removed from lists, which affects the ranks of the remaining items.

Next we review the remaining local variables used by processors. Variable \( \text{Phase}_v \) contains the name of the current phase performed by the processor, which is either Main or Closing. Processor \( v \) maintains variable \( \text{Done}_v \), which remains false until \( v \) learns that all the tasks have been performed. As long as the variable remains false, processor \( v \) is busy. Variable \( \text{Selected}_v \) is used to store a name of a processor to whom Stop signal is to be sent in the current instance of closing phase.

The information stored in the three lists at a processor represents local knowledge and may be out of date. In the case of lists \( \text{Processors}_v \) and \( \text{Busy}_v \), the sets stored in them may be supersets of the true sets of active and busy processors. This happens if some processors are either not active or not busy, but processor \( v \) has not learned this yet. Similarly, the set of task names on list \( \text{Tasks}_v \) may be a superset of the actual set of the remaining tasks, since some of the tasks may have been performed but processor \( v \) does not know this yet.

### Selecting items from lists

Processors executing the algorithm repeatedly select items from their lists. For instance, a busy processors selects a task to perform from its local copy of list \( \text{Tasks} \) while executing the main phase. Similarly, a processor that is not busy selects a processor from its local copy of list \( \text{Busy} \) to send signal to stop executing the main phase. We define two selection rules, one deterministic and the other randomized.

The deterministic rule \( \text{Load-Balance} \) assigns items by load balancing. Let \( q \) be the number of elements in the list at hand. Define the rank \( r(v) \) of \( v \) as the largest integer satisfying the inequalities

\[
q \cdot \frac{v}{p} < r(v) < q \cdot \frac{v}{p} + 1.
\]  

Processor \( v \) selects the item at position \( r(v) \) on the list, where the positions are numbered starting from 1. Observe that if \( q > p \), then each processor is assigned a different task, otherwise each task is assigned to at least one processor.

Randomized rule \( \text{Select-Randomly} \) is defined as follows:

- if the size of the list, for which the rule is applied, is at least \( 11p^2g(p) \), then the rule operates in the same way as \( \text{Load-Balance} \),
- otherwise an item is selected uniformly at random, with selections being independent over different rounds and different processors.
Updating knowledge. Every processor \( v \) updates its local variables after collecting all the messages in the first round of a phase. When a message brings a copy of a list to \( v \), then \( v \) compares the copy in the message with its local one. The recipient removes all the entries from the local copy of the list that are not present in the list brought by the message. If no message is received from a neighboring processor \( u \), then \( u \) is removed from the list \( \text{Processors}_v \). Finally, the received messages may affect the status of processor \( v \) as being busy executing the main phase. If a message sent to \( v \) brings an instance of \( \text{Busy} \) with some processors missing in it, then \( v \) immediately sets \( \text{Done}_v \) to true. If the \( \text{Stop} \) signal is received by \( v \), then \( v \) sets \( \text{Done}_v \) to true and removes \( u \) from the list \( \text{Processors}_v \). Finally, a recipient of an empty copy of \( \text{Tasks} \) stops being busy and switches to the closing phase.

A processor \( v \) estimates the size of its range when performing the closing phase. To this end, \( v \) considers the subgraph of \( L(p)^\ell \) induced by the processor in its local list \( \text{Processors}_v \) to be the current working part of the communication graph \( G(p) \). The distances from all nodes of \( G(p) \) to \( v \) are computed by \( v \) in this subgraph of \( G(p) \). If the size of the range of processor \( v \) obtained this way is at least \((p-f)/7\), then \( v \) is said to believe itself compact. The distances computed by \( v \) are underestimates, therefore processor \( v \) may believe itself compact while in reality it is not.

**Lemma 2.** Algorithm Randomized-DoAll is a solution of Do-All.

**Proof.** The algorithm is an instantiation of a generic algorithm given in [2]. It is shown in [2] that any such instantiation, obtained by specifying a rule to select items from lists, is a correct Do-All solution. \( \square \)

Communication complexity. The following Lemma 3 gives a bound on the expected communication cost of Randomized-DoAll, in terms of the expected work and the magnitude of the number \( f < p \) of crashes.

**Lemma 3.** If the expected work complexity of algorithm Randomized-DoAll is \( W \), then the expected communication cost of algorithm Randomized-DoAll is

\[
\mathcal{M} = O(W \cdot \Delta) = O\left(W \cdot \min\left\{p, \left(\frac{p}{p-f}\right)^{3.4}\right\}\right).
\]

**Proof.** Algorithm Randomized-DoAll uses graph \( L(p)^\ell \) as the communication graph, where \( \ell \) is such that Lemma 1 holds. A processor multicasts \( \min(\Delta, p) \) messages to its neighbors in the communication graph during each phase. Direct calculations show that the estimate of the maximum degree of a node in the communication graph given in Lemma 1 for \( \Delta_0 = 74 \) and \( \rho = 27/2 \) becomes \( \Delta = O((p-f)^{3.4}) \). Every processor performing the closing phase sends a single \( \text{Stop} \) message at the end of the phase. The total number of these messages is clearly bounded above by the work accrued during closing phases, which is \( O(W) \). \( \square \)

Observe that the code of algorithm Randomized-DoAll is explicit, in the sense that it can be completely specified in time polynomial in \( p \), given \( p, t \) and \( f < p \). This follows from the explicitness of the construction of \( L(p) \) and from Lemma 1. Randomness of the algorithms does not affect its explicitness, in the sense that we make the standard assumption that a perfect source of randomness is available as a black box.

5. Analysis of complexity

We partition an execution into disjoint epochs, where epoch \( E_i \) is defined to be the \( i \)th segment of \( g(p) = 30\log_2 p + 2 \) consecutive phases. An epoch is of sufficient duration for all the processors in a compact subgraph to communicate among themselves.

The notation \( G_i \) means a subgraph of the communication graph \( G(p) \) induced by these vertices that are active during the first phase of epoch \( E_i \). Clearly \( G_{i+1} \subseteq G_i \) for \( i \geq 1 \). We use notation \( H_i \) to denote a compact subgraph of \( G_i \) such that \( H_{i+1} \subseteq H_i \), for \( i \geq 1 \). These graphs are conceptual in that they are considered for the sake of analysis only. They exist by the subgraph property, as explained in Section 3. Graphs \( G_i \) and \( H_i \) depend on how failures occur in an execution. Our analysis is based on just one possible descending sequence \( \{H_i\}_{i \geq 1} \) of compact subgraphs of active nodes in \( G(p) \) in an execution; this sequence is not meant to be unique but to indicate specific graphs to refer to.

The following notation is used in this section. Let \( T_{v,i} \) be the set of elements in \( \text{Tasks}_v \) at the beginning of the first phase of \( E_i \). Let \( U_i \) stand for \( \bigcup_{v \in E_i} T_{v,i} \) and number \( u_i \) for \(|U_i|\). When the index \( i \) of epoch \( E_i \) is understood from context, we drop the subscript \( i \) in \( T_{v,i} \) and write simply \( T_v \).

The analysis we carry out in detail is for the case when the processors perform main phases. When processors perform the closing phase, then list \( \text{Busy} \) plays the role of \( \text{Tasks} \) and sending \( \text{Stop} \) signal to a processor is interpreted as performing the task to stop the processor from being active. An analysis for the main phase resulting in a bound on work can be translated into an analysis of work performed during closing phases by taking a bound for main phases in terms \( p \) and \( t \) and plugging in \( p \) for \( t \).

The following Lemma 4 was used in [2] to assess the performance of such an instantiation of the generic algorithm in which selecting from lists is based solely on deterministic load-balancing.
Lemma 4. (See [2].) If \( u_i \geq 11 p^2 g(p) \) and only the selection rule **Load-Balance** is used in an execution, then \( u_i - u_{i+1} \geq |H_{i+1}|g(p) \).

To analyze the work accrued while tasks are selected randomly, we consider a modification of the processes of task selection and communication as it is specified in the pseudo-code of the algorithm. This is done in order to simplify the derivations of estimates on the probabilities of deviation of random variables from their expected values. Algorithm **RANDOMIZED-DOALL** operates in such a way that once a task has been performed, then it is immediately removed from the list **Tasks** at the processor, which in turn triggers a process of proliferation of the information that the task has been performed among all the processors. This intertwining of task completion and communication improves performance but hinders analysis, because the events of a specific task being performed by two different processors are not independent. The nodes in graphs \( H_i \) receive messages from outside of \( H_i \), and this again improves the performance but complicates the analysis. For the sake to simplify the analysis, we separate work from communication within each epoch. We estimate the progress made by the processors in graphs \( H_i \) by relying only on the work by the nodes in \( H_i \). This means we treat graphs \( H_i \) as ‘isolated worlds’ by ignoring in the analysis any messages arriving from outside so that only the communication within \( H_i \) affects the progress made. Details are as follows.

As we analyze a modified algorithm, the technical lemmas are formulated for the original algorithm. The modified algorithm is conceptual and considered for the purpose of analysis only. This modified algorithm has each single original epoch replaced by two consecutive epochs: the first conceptual epoch is devoted entirely to performing tasks but no messages are exchanged, while in the next conceptual epoch only messages are exchanged but tasks are not performed at all. The rules governing selecting tasks and processing messages are exactly the same as in the pseudo-code of **RANDOMIZED-DOALL** algorithm. The same graph \( H_i \) is used in the analysis, for each of these two epochs corresponding to the \( i \)th epoch \( \mathcal{E}_i \) of the original algorithm. We treat each graph \( H_i \) in isolation from the remaining part of the communication graph, in that we ignore messages bringing any information from outside. We do not introduce new notation to refer to the modified algorithm. In particular, it is sufficient to refer simply to epoch \( \mathcal{E}_i \), since we can specify which corresponding conceptual epoch we mean by the context.

This approach to the analysis hinges on the following simple observation: if the modified algorithm is shown to perform all the tasks with some amount of work and communication, then these same performance bounds hold true for the original algorithm. The processors do not communicate among themselves at all in the first conceptual epoch, therefore we can interpret their behavior by random variables that are independent. The second conceptual epoch needs not to be analyzed at all: we simply rely on the property that an epoch is of a sufficient duration for all the processors in a compact subgraph to communicate among themselves. Therefore in the first conceptual epoch some progress is made by each processor independently of what the other processors do, and in the second conceptual epoch the processors exchange information about the progress made to make the set of tasks still outstanding a common knowledge. In the following Lemmas 5–7, notation \( \mathcal{E}_i \) is to be interpreted as referring to the first conceptual epoch, corresponding to the \( i \)th epoch \( \mathcal{E}_i \) of the original algorithm, during which tasks are performed but there is no communication.

We may interpret the selection of tasks as throwing ball into bins, with a task understood as a ‘bin’ and a selection as a ‘ball’. Such selections by the same processor during an epoch are not independent, as a task selected once will not be selected again. To simplify the analysis, we will consider random selection without immediately removing each selected task, so that a task may be selected by the same processor multiple times, which still gives sufficient estimates. More precisely, we consider a modification of the algorithm in which a selected task is performed immediately but it is removed from the list of outstanding tasks only at the end of an epoch.

We will refer to experiments in which \( m \) balls are placed independently and uniformly at random into \( k \) bins. Let \( L \) be the number of empty bins. The following is an estimate on the expected value of \( L \):

\[
E[L] = k \left( 1 - \frac{1}{k} \right)^m \leq ke^{-m/k}.
\]

The probability of deviating from this expected value is estimated by the inequality

\[
\Pr\{ |L - E[L]| \geq \varepsilon \} \leq 2e^{-2\varepsilon^2/m},
\]

which can be shown by the Azuma-Hoeffding inequality, see unit 12.5.3 in [15].

Lemma 5. If \( u_i \geq |H_i|g(p)/2^6 \), then either more than \(|H_i|/2 \) processors fail during epoch \( \mathcal{E}_i \) or \( u_i - u_{i+1} \geq |H_i|g(p)/2^3 \) with \( 1 - \exp(-\Omega(|H_i| \log p)) \) probability.

**Proof.** Suppose that at least \(|H_i|/2 \) processors in \( H_i \) survive through epoch \( \mathcal{E}_i \). Let \( H \subseteq H_{i+1} \) denote an arbitrary induced compact subgraph such that \(|H| = |H_i|/2\). Denote the number \(|H_i|g(p)/2 \) by \( \alpha \); this precisely is the number of choices of tasks made by all the processors in \( H \) during epoch \( \mathcal{E}_i \). Notice that the assumption of the lemma reads \( u_i \geq \alpha/2^5 \). Our first goal is to show that the processors in \( H \) remove less than \( \alpha/2^7 \) tasks from all their lists with \( \exp(-\Omega(|H_i| \log p)) \) probability.

We interpret the tasks, outstanding at the beginning of the epoch, as bins. Selecting a task represents placing a ball in the respective bin. After the epoch is over, empty bins represent the tasks that have not been performed in the epoch. Therefore
The inequality is the second form of the bound, for any \( m \) which follows from (2) by the additional assumptions we have. To this end, consider two ranges of \( E \) during epoch \( \mathcal{E}_i \) while \( u_{i+1} \) is a random variable. We use \( L \), in a meaning consistent with (2), to denote the number of tasks not performed during epoch \( \mathcal{E}_i \), but we also have \( L = u_{i+1} \). We first show the estimate
\[
E[L] \leq \max\{ke^{-3/4}, k - m/4\},
\] which follows from (2) by the additional assumptions we have. To this end, consider two ranges of \( m \) with respect to \( k \). When \( 3k/4 \leq m \leq 2^5k \), then the inequality \( ke^{-m/k} \leq ke^{-3/4} \) holds, by interpreting \( ke^{-m/k} \) as a decreasing function of \( m \). The estimate \( E[L] \leq k - m/4 \) holds for the range \( m < 3k/4 \). To see this, visualize the process of throwing \( m \) balls into \( k \) bins as a sequence of steps, a step per ball, and observe that the probability that a given ball hits an empty bin is always larger than 1/4. It follows, by the linearity of expectation, that the expected number of nonempty bins is at least \( m/4 \).

Translate (4) into the notation in which the lemma is formulated to obtain
\[
E[u_{i+1}] \leq \max\{u_i e^{-3/4}, u_i - \alpha/4\},
\] which implies
\[
u_{i+1} - E[u_{i+1}] \geq u_{i+1} - \max\{u_i e^{-3/4}, u_i - \alpha/4\}.
\] The bound
\[
Pr[u_{i+1} - E[u_{i+1}] \geq \alpha/2^7] \leq 2e^{-2(\alpha/2^7)^2/\alpha}
\] is obtained from (3) after a similar translation of notation. Combine (5) with (6) to obtain
\[
Pr[u_{i+1} - \max\{u_i e^{-3/4}, u_i - \alpha/4\} \geq \alpha/2^7] \leq 2e^{-\alpha/2^{15}}.
\] This together with the inequality \( u_i \geq \alpha/2^5 \) gives
\[
Pr[u_i - u_{i+1} \leq \alpha/2^7] \leq Pr[u_i - u_{i+1} \leq \max\{u_i/2, \alpha/4\} - \alpha/2^7]
\leq Pr[u_i - u_{i+1} \leq \max\{u_i(1 - e^{-3/4}), \alpha/4\} - \alpha/2^7]
= Pr[u_{i+1} - \max\{u_i e^{-3/4}, u_i - \alpha/4\} \geq \alpha/2^7]
\leq 2e^{-\alpha/2^{15}}.
\] which is \( \exp(-\Omega(\alpha)) \).

Next we consider all the possible subgraphs \( H \). There are at most
\[
\left(\frac{|H|}{|H|/2}\right) \leq 2^{|H|}
\] subsets of processors that can induce \( H \). Whichever \( H \) contributes to performing outstanding tasks in epoch \( \mathcal{E}_i \), the available non-faulty processors remove at least \( \alpha/2^7 \) tasks from their initial lists of outstanding tasks with probability
\[
1 - \sum_H \exp(-\Omega(\alpha)) \geq 1 - 2^{|H|} \exp(-\Omega(\alpha))
\geq 1 - \exp(-\Omega(|H| \log p))
\] which is the bound we wanted to prove. □

We use Chernoff bounds on the probability of deviation of a sum of independent random variables from its expectation, see [15] for a detailed exposition. Let \( 0 < r < 1 \), let \( X_1, \ldots, X_n \) be a sequence of Bernoulli trials with \( Pr[X_j = 1] = r \) the probability of success and \( Pr[X_j = 0] = 1 - r \) the probability of failure, for each \( j \). Let \( S = \sum_{j=1}^n X_j \) and let \( \mu = ES \) be the expected value of \( S \). We have \( \mu = nr \). The first form of the bound is as follows. For any \( 0 < \epsilon < 1 \), we have
\[
Pr[S \leq (1 - \epsilon)\mu] \leq \exp(-\mu\epsilon^2/2).
\] The inequality
\[
Pr[S \geq b\mu] \leq \exp(\mu(b - 1 - b \ln b))
\] is the second form of the bound, for any \( b > 1 \).
Lemma 6. Assume the inequalities $2|H_i| \leq u_i < |H_i|g(p)/Z^6$. Then either more than $|H_i|/3$ processors fail during the epoch $E_i$ or the inequality
\[
\frac{u_{i+1}}{u_i} \leq \frac{|H_i|}{|H_i|g(p)}
\]
holds with $1 - \exp(-\Omega(|H_i|))$ probability.

Proof. Let $Z \subseteq \bigcup_{v \in H_i} T_{v,i}$ be an arbitrary subset of size $\frac{u^2_i}{|H_i|g(p)}$. The probability that processor $v$ does not perform a task from $Z$ during epoch $E_i$ is at most
\[
\left(1 - \frac{|Z|}{u_i}\right)^{g(p)} \leq \exp\left(-\frac{|Z|}{u_i} \cdot g(p)\right) = \exp\left(-\frac{u_i}{|H_i|}\right).
\]
Let $\beta$ stand for $\exp\left(\frac{u_i}{|H_i|}\right)$. Given $v \in H_i$, let $X_v = 1$ if in epoch $E_i$ a task from $Z$ has not been performed by processor $v$, and $X_v = 0$ otherwise. Define $X = \sum_{v \in H_i} X_v$. We have $\Pr[X_v = 1] \leq 1/\beta$ and $\mathbb{E}[X] \leq |H_i|/\beta$.

By the assumption that no messages are sent during performing tasks, random variables $X_v$ can be assumed to be independent for the sake of the analysis. By the Chernoff bound (8) and by the inequality $u_i \geq 2|H_i|$, we obtain the following estimates:
\[
\Pr\left[X > \frac{2}{3}|H_i|\right] = \Pr\left[X > \left(\frac{|H_i|}{\beta}\right) \cdot \frac{2\beta}{3}\right] = \Pr\left[X > \mathbb{E}[X] \cdot \frac{2\beta}{3}\right] < \exp\left(\mathbb{E}[X] \cdot \left(\frac{2\beta}{3} - 1 - \frac{2\beta}{3} \log \frac{2\beta}{3}\right)\right) < \exp\left(\mathbb{E}[X] \cdot \left(\frac{2}{3} \exp\left(-\frac{u_i}{|H_i|}\right) - 1 - \frac{u_i}{3|H_i|} \exp\left(-\frac{u_i}{|H_i|}\right)\right)\right) \leq \exp\left(|H_i| \exp\left(-\frac{u_i}{|H_i|}\right) \cdot \left(1 - \frac{u_i}{3|H_i|} \exp\left(-\frac{u_i}{|H_i|}\right)\right)\right) = \exp(-u_i/3). \tag{9}
\]

The number of all such subsets $Z$ can be estimated as follows:
\[
\left(\frac{u_i}{u^2_i/(|H_i|g(p))}\right) \leq 2^{h(x)} \leq 2^{h(u_i/2^{-6})} \leq \exp(u_i/3\ln 2). \tag{10}
\]
where $h(x) = -[x\log_2 x + (1 - x)\log_2(1 - x)]$ is the (binary) entropy function and $h(2^{-6}) \leq 1/3$.

Consider the event that for each subset $Z$ of size $u^2_i/(|H_i|g(p))$ at least $|H_i|/3$ processors perform some task in $Z$ each. The probability of this is at least
\[
1 - \exp(-u_i/3) \exp\left((u_i/3)\ln 2\right) \geq 1 - \exp\left((u_i/3)(-1 + \ln 2)\right) \geq 1 - \exp\left(-u_i/12\right) \geq 1 - \exp(-\Omega(|H_i|)),
\]
by bounds (9) and (10), because $u_i \geq 2|H_i|$. We have thus shown that, with $1 - \exp(-\Omega(|H_i|))$ probability, the only possibility for the adversary to preserve $u_{i+1} > u^2_i/(|H_i|g(p))$ tasks for the next epoch $E_{i+1}$ is to fail at least $|H_i|/3$ processors. □

Lemma 7. If $g(p) \leq u_i < 2|H_i|$, then either more than $|H_i|/4$ processors fail during epoch $E_i$ or $u_{i+1} \leq u_i/g(p)$ with $1 - \exp(-\Omega(|H_i|))$ probability.

Proof. Let $Z$ be a set of $u_i/g(p)$ tasks in $U_i$. First we show that at least $|H_i|/4$ processors in $H_i$ perform some tasks in $Z$ each with $1 - \exp(-\Omega(|H_i|))$ probability. Fix one processor, say $v$, in $H_i$. The probability that processor $v$ will not perform a task in $Z$ is at most
\[
\left(1 - \frac{|Z|}{u_i}\right)^{g(p)} = \left(1 - \frac{1}{g(p)}\right)^{g(p)} < 1/e.
\]
For any processor \( v \in H_i \), let random variable \( X_v \) equal 1 if processor \( v \) performs a task in \( Z \) during \( \mathcal{E}_i \), and be 0 otherwise. Define \( X = \sum_{v \in H} X_v \) and let \( \mu = \mathbb{E}[X] \) be the expected value of \( X \). We have \( \Pr[X_v = 1] > 1 - 1/e \) and \( \mu > (1 - 1/e)|H_i| \). The random variables \( X_v \) can be assumed to be independent, for the sake of the analysis, by the assumption that no messages are sent during performing tasks. By the Chernoff bound (7) we obtain
\[
\Pr\left[ \sum_{v \in H_i} X_v < |H_i|/4 \right] \leq \Pr\left[ \sum_{v \in H_i} X_v < (1 - 1/e)|H_i|/2 \right] 
\leq \Pr\left[ \sum_{v \in H_i} X_v < \mu/2 \right] 
< \exp(-\mu/12) < \exp(-|H_i|/24).
\]

The number of subsets \( Z \) is
\[
\binom{u_i}{u_i/g(p)} \leq 2^{u_i h(1/g(p))} < 2^{2|H_i| h(1/g(p))}.
\]

Consider the event when, for each set \( Z \) of \( u_i/g(p) \) tasks, there exist \( |H_i|/4 \) processors such that each of them performs some task in \( Z \) during epoch \( \mathcal{E}_i \). This event holds with probability at least
\[
1 - \sum_{z} \exp(-|H_i|/24) \geq 1 - \exp(-|H_i|/24) \cdot 2^{2|H_i| h(1/g(p))} 
> 1 - \exp(-\Omega(|H_i|)),
\]

because \( h(x) \) converges to zero, when \( x \) tends to zero, and function \( g(p) \) grows arbitrarily large, as \( p \) grows to infinity. If this event occurs, then the only possibility for the adversary to preserve more than \( u_i/g(p) \) tasks, for the next epoch \( \mathcal{E}_{i+1} \), is to fail at least \( |H_i|/4 \) processors that want to perform tasks during the epoch. □

For a given subgraph \( H \subseteq G(p) \), let \( A_H \) be the set of epochs \( \mathcal{E}_i \) such that \( H \subseteq H_i \) and \( |H_i| < 2|H| \). Let \( a_H \) be the number of the first epoch \( \mathcal{E}_i \) in \( A_H \), and let \( b_H \) be the number of the first epoch that is not in \( A_H \) and is greater than \( a_H \). Our next goal is to estimate \( b_H - a_H \).

We partition the numbers of epochs in \( A_H \) into the following five disjoint intervals \( I_j \), \( 1 \leq j \leq 5 \), in such a way that if \( x \in I_{j_1} \) and \( y \in I_{j_2} \), for \( j_1 < j_2 \), then \( x < y \):
- Interval \( I_1 \) contains these \( i \in A_H \) for which \( u_i \geq 11p^2 g(p) \).
- Interval \( I_2 \) contains these \( i \in A_H \) for which \( |H_i| g(p)/2^6 \leq u_i < 11p^2 g(p) \).
- Interval \( I_3 \) contains these \( i \in A_H \) for which \( 2|H_i| \leq u_i < |H_i| g(p)/2^6 \).
- Interval \( I_4 \) contains these \( i \in A_H \) for which \( g(p) \leq u_i < 2|H| \).
- Interval \( I_5 \) contains these \( i \in A_H \) for which \( 1 \leq u_i < g(p) \).

Observe that \( b_H - a_H = \sum_{j=1}^5 |I_j| \). Next we estimate the sizes of these intervals. By Lemma 4, during epochs in \( I_1 \) all the processors in \( H \) perform different tasks, hence the estimate
\[
|I_1| = \mathcal{O}\left(1 + \frac{u_{b_H} - u_{b_H}}{|H| g(p)}\right) \tag{11}
\]
holds with probability 1.

The number of epochs in \( I_2 \) is estimated by Lemma 5. The property \( |H_{i+1}| < |H_i|/2 \) holds only in \( \mathcal{O}(1) \) epochs in \( I_2 \), because of the definition of graph \( H \) and set \( A_H \). Consider the remaining epochs in \( I_2 \). By Lemma 5, the total number of epochs in \( I_2 \) is
\[
|I_2| = \mathcal{O}\left(1 + \frac{u_{b_H} - u_{b_H}}{|H| g(p)}\right), \tag{12}
\]
with \( 1 - \exp(-\Omega(|H| \log p)) \) probability. This is because, by the definition of \( I_2 \), we have the estimate \( \frac{u_{b_H} - u_{b_H}}{|H| g(p)} < p^2 \), and hence the probability of progress in all the epochs in \( I_2 \) can be bounded below by
\[
1 - \mathcal{O}\left(\frac{u_{b_H} - u_{b_H} + 1}{|H| \log p}\right) \exp(-\Omega(|H| \log p)) \geq 1 - \mathcal{O}(p^2) \exp(-\Omega(|H| \log p)) 
\geq 1 - \exp(-\Omega(|H| \log p)),
\]
if \( |H| \) is sufficiently large.
The size of interval $I_3$ is estimated by Lemma 6. We may assume $|I_3| > 2$. Notice that the property $|H_{i+1}| < 2|H_i|/3$ holds only in $O(1)$ epochs in $I_3$. Consider the remaining epochs in $I_3$. Let $k$ be the largest epoch number in $I_3$, and let $m = |I_3|$. Iterating the recursive estimate $u_{i+1} \leq \frac{u_i^2}{|H_i|^2}$ yields the following sequence of inequalities:

$$2|H| \leq u_k \leq \frac{u_k^2}{|H_i|^2(g(p))} \leq \frac{u_k^4}{|H_i|^4(g(p))^3} \leq \cdots \leq \frac{u_k^{2m-1}}{|H_i|^{2m-1}(g(p))^{2m-1}}.$$ 

Since $u_{k-m+1} < |H_i|^2(g(p))/2^6$, we obtain that

$$u_k \leq \frac{u_k^{2m-1}}{|H_i|^{2m-1}(g(p))^{2m-1}} < \frac{u_{k-m+1}}{|H_i|^2(g(p))^{2m-1}} \leq \frac{u_{k-m+1}}{2^{2m-1}}.$$ 

and it follows that the inequality

$$2^{2m-1} < \frac{u_{k-m+1}}{u_k}$$

holds. Because of the estimate

$$\frac{u_{k-m+1}}{u_k} \leq \min \left\{ g(p), \frac{u_{a_H}}{u_{b_H}} \right\},$$

we obtain that

$$|I_3| = m + O(1) = O(1 + \log \log g(p)) \quad (13)$$

with probability

$$1 - m \cdot \exp(-\Omega(|H|)) \geq 1 - O(\log \log g(p)) \cdot \exp(-\Omega(|H|)),$$

which is $1 - \exp(-\Omega(|H|))$ for $|H| = \omega(\log \log g(p))$.

The epochs in $I_4$ fall in the scope of applicability of Lemma 7. It follows from the definition of $I_4$ that $|H| > g(p)/2$. The property $|H_{i+1}| < 3|H_i|/4$ holds only in $O(1)$ epochs in $I_4$. Consider the remaining epochs in $I_4$. Let $k$ be the largest epoch number in $I_4$, and let $n = |I_4|$. We have

$$g(p) \leq u_k \leq \frac{u_k^{2m-1}}{g(p)} \leq \cdots \leq \frac{u_k^{2m-1}}{g(p)^{n-1}}.$$

which implies $g(p)^{n-1} \leq u_{k-n+1}/u_k$. Combine this with the bound

$$\frac{u_{k-n+1}}{u_k} \leq \min \left\{ \frac{2|H|}{g(p)}, \frac{u_{a_H}}{u_{b_H}} \right\}$$

to obtain

$$|I_4| = n + O(1) = O \left( 1 + \frac{\log u_{k-n+1}}{\log g(p)} \right) = O \left( 1 + \frac{\log u_{a_H}}{\log g(p)} \right) \quad (14)$$

with probability

$$1 - n \cdot \exp(-\Omega(|H|)) \geq 1 - \log \frac{2|H|}{g(p)} \cdot \exp(-\Omega(|H|)),$$

which is $1 - \exp(-\Omega(|H|))$ when $|H| > g(p)$.

The number of epochs in $I_5$ is $O(1)$ with probability 1, since every node in $H$ has less than $g(p)$ tasks to perform, which can be achieved during one epoch.

Lemma 8. Suppose $p - f > g(p)$. There is a constant $\alpha > 0$ such that, for any compact subgraph $H \subseteq G(p)$ of size at least $p - f$, if no processors in $H$ ever crash, then

$$b_H - a_H \leq \alpha \cdot \left( 1 + \frac{u_{a_H}}{u_{b_H}} + \log \log p + \frac{\log u_{a_H}}{\log g(p)} \right)$$

with $1 - \exp(-\Omega(|H|))$ probability.

Proof. The assumption $p - f > g(p)$ implies $|H| > g(p)$, which makes estimates (11), (12), (13) and (14) hold with $1 - \exp(-\Omega(|H|))$ probability. This means that

$$\sum_{j=1}^{5} |I_j| = O \left( \frac{u_{a_H}}{u_{b_H}} + 1 \right) + O(\log \log p) + O \left( \frac{\log u_{a_H}}{\log \log p} \right) + O(1) \quad \text{with} \quad 1 - \exp(-\Omega(|H|)) \text{ probability.} \quad \Box$$
Lemma 9. Suppose $|H| \leq g^2(p)$ and there are at least $u_{bh} > \frac{p}{\log \log p}$ outstanding tasks at the beginning of epoch $b_H$. There is a constant $\alpha > 0$ such that, for any compact subgraph $H \subseteq G(p)$ of size at least $p - f$, if no processors in $H$ ever crash, then

$$b_H - a_H \leq \alpha \cdot \left( 1 + \frac{u_{bh} - u_{bu}}{|H|g(p)} \right)$$

with $1 - \exp(-\Omega(|H| \log p))$ probability.

Proof. Intervals $I_3$, $I_4$ and $I_5$ are empty under these assumptions, for sufficiently large $p$. We use estimates (11) and (12), which hold with $1 - \exp(-\Omega(|H| \log p))$ probability. We obtain

$$b_H - a_H = |I_1| + |I_2| = O \left( 1 + \frac{u_{bh} - u_{bu}}{|H|g(p)} \right)$$

with $1 - \exp(-\Omega(|H| \log p))$ probability. \qed

Theorem 1. For any $a > 1$, algorithm Randomized-DoAll solves the Do-All problem with

$$\mathcal{W} = O \left( t + p \frac{\log p}{\log \log p} \right)$$

work with probability at least $1 - p^{-a}$ against the $f$-bounded adversary, for sufficiently large $p$ and any $f < p$.

Proof. We first consider the main phases. We partition the epochs $\mathcal{E}_i$ into groups depending on the size of graphs $H_i$. Epoch $\mathcal{E}_i$ is said to be in group $S(\ell)$ if $2^\ell \leq |H_i| < 2^{\ell + 1}$, for $\ell < \log_2 p$. Let $\tilde{H}_\ell$ be the smallest graph in group $S(\ell)$.

Let $\tilde{u}_\ell$ denote $u_i$ for $i$ such that $\mathcal{E}_i$ is the first epoch in group $S(\ell)$. Let $\tilde{u}_f$ be the first epoch after all the epochs in group $S(\ell)$. An execution is partitioned into two parts, depending on the magnitude of $\ell$.

Case 1: Groups $S(\ell)$ when $\ell > 2 \log g(p)$.

We apply Lemma 8, where $\tilde{H}_\ell$ is taken as $H$, group $S(\ell)$ is taken as the set $A_H$, $\tilde{u}_\ell$ as $a_H$, and $\tilde{u}_f$ as $b_H$. We obtain that the amount of work during these epochs is bounded above by

$$\sum_{\ell=2^{\log g(p)}}^{\log p} |\tilde{H}_\ell|g(p) \cdot \alpha \left( 1 + \frac{\tilde{u}_\ell - \tilde{u}_f}{|\tilde{H}_\ell|g(p)} + \log \log p + \frac{\log \tilde{u}_f}{\log p} \right)$$

$$= O \left( \sum_{\ell=2^{\log g(p)}}^{\log p} \left( \tilde{u}_\ell - \tilde{u}_f + |H_i|g(p) \left( 1 + \log \log p + \frac{\log \tilde{u}_f}{\log p} \right) \right) \right)$$

$$= O \left( t + g(p) \log \log p + \sum_{\ell=2^{\log g(p)}}^{\log p} 2^\ell \log g(p) + \sum_{\ell=2^{\log g(p)}}^{\log p} \log \tilde{u}_f \log p \right)$$

$$= O \left( t + p \log p \log \log p + p \log p \log t \log \log p \right)$$

$$= O \left( t + \frac{p \log^2 p}{\log \log p} \right),$$

because $\log t = O(\log p)$ for $t = O(p \log \log p)$. This estimate holds true with probability

$$1 - \sum_{\ell=2^{\log g(p)}}^{\log p} \exp(-\Omega(|\tilde{H}_\ell|)) \geq 1 - \sum_{\ell=2^{\log g(p)}}^{\log p} \exp(-\Omega(2^\ell))$$

$$\geq 1 - \exp(-\Omega(\log^2 p)),$$

which is $1 - n^{-a}$ for sufficiently large $p$.

Case 2: Groups $S(\ell)$ when $\ell \leq 2 \log g(p)$.

If the number of outstanding tasks is smaller than $\frac{p}{\log \log p}$, then the work accrued when every active processor performs all the remaining tasks is

$$\mathcal{W} = O \left( \sum_{\ell=0}^{2^{\log g(p)}} |H_i| \cdot \frac{p}{\log \log p} \right) = O(p \log p).$$
Let \( \gamma > 0 \) be a fixed integer parameter, of magnitude to be determined later. If \(|H| < \gamma\), then the work accrued when every active processor performs all the remaining tasks is \( O(t)\). What remains is the case when \(|H| \geq \gamma\) and the number of outstanding tasks is at least \( p \frac{\log g(p)}{\log \log p} \). We obtain that the work performed during these epochs is

\[
O\left( \sum_{\ell=0}^{2\log g(p)} |H| g(p) \left( 1 + \frac{\bar{H}_\ell - U_{\ell}}{|H| g(p)} \right) \right) = O(t + p),
\]

by Lemma 9. This holds with probability

\[
1 - \sum_{\ell=\gamma}^{2\log g(p)} \exp(-\Omega(|H| \log p)) \geq 1 - \sum_{\ell=\gamma}^{2\log g(p)} \exp(-\Omega(\gamma \log p)),
\]

which is \( 1 - n^{-a} \), for a sufficiently large \( \gamma > 0 \).

The closing phases are tackled similarly, with stopping processors interpreted as tasks. Replace \( t \) with \( p \) in the estimate \( O(t + p \frac{\log^2 p}{\log \log p}) \) to obtain \( O(p + p \frac{\log^2 p}{\log \log p}) \), which is \( O(t + p \frac{\log^2 p}{\log \log p}) \).

**Corollary 1.** The expected work of algorithm Randomized-DoAll is \( O(t + p \frac{\log^2 p}{\log \log p}) \).

**Proof.** By Theorem 1, the work of the algorithm is \( O(t + p \frac{\log^2 p}{\log \log p}) \) with probability at least \( 1 - p^{-a} \). The work accrued when every processor performs every tasks is \( O(pt) \). Combine these two facts to obtain \( O((1-p^{-a})(t + p \frac{\log^2 p}{\log \log p} + p^{-a} pt)) \) expected work, which is \( O(t + p \frac{\log^2 p}{\log \log p}) \) for a sufficiently large \( a > 1 \).

**Corollary 2.** The expected effort of algorithm Randomized-DoAll is \( O(t + p \frac{\log^2 p}{\log \log p}) \) against an adaptive linearly-bounded adversary.

**Proof.** By Corollary 1 and Lemma 3 the expected communication is

\[
O\left( \left( t + p \frac{\log^2 p}{\log \log p} \right) \cdot \min\left\{ p, \left( \frac{p}{p - f} \right)^{3.4} \right\} \right).
\]

If the adversary is linearly bounded, then \( \left( \frac{p}{p - f} \right)^{3.4} = O(1) \), and so the expected communication is of the same order of magnitude as the expected work.

### 6. Discussion

The expected work \( O(p \log^2 p / \log \log p) \) of the algorithm given in this paper, for any \( f < p \) and the case of \( t = O(p) \), is asymptotically smaller by the \( \log \log p \) factor than the worst-case estimate on work of the deterministic algorithm given in [2]. Finding the optimum expected complexity of a randomized solution of Do-All is an open problem. It is not known if randomization affects the complexity of Do-All against the unbounded adversary. It is shown in [3] that there is a randomized solution to the Do-All problem with a smaller expected work complexity than the worst-case work of any deterministic solution against weakly-adaptive linearly-bounded adversaries.

The work accrued by any Do-All solution has to be \( \Omega(p \log p / \log \log p) \) for \( t = p \) against the unbounded adversary, in both deterministic and randomized cases, see [3,11]. A deterministic algorithm that schedules tasks by instantaneous perfect balancing in each round performs \( O(p \log p / \log \log p) \) work against linearly-bounded adversaries, see [3], hence \( \Theta(p \log p / \log \log p) \) work is optimal in such a setting. The communication overhead to keep the load perfectly balanced throughout an execution may be inherently large. This paper, following [2], explores solving Do-All in ways that are efficient in terms of both work and communication. In particular, if the adversary is linearly bounded, the algorithm presented in this paper accrues the amount of work that is asymptotically equal to the number of point-to-point messages.

### Acknowledgement

We thank the anonymous reviewers for helping to remove flaws from the initial versions of some of the proofs.

### References

