Ordered Balancing: Load Balancing for Redundant Task Scheduling in Robotic Network Cloud Systems

Saeid[†]Alirezazadeh[1*] and Luís A. Alexandre[2]

[1*]C4 - Cloud Computing Competence Centre (C4-UBI), Universidade da Beira Interior, Covilhã, Portugal.
[2]NOVA LINCS, Universidade da Beira Interior, Covilhã, Portugal.

*Corresponding author(s). E-mail(s): saeid.alirezazadeh@gmail.com;
Contributing authors: luis.alexandre@ubi.pt;

Abstract

To perform a set of tasks in a robotic network cloud system as fast as possible, it is recommended to use a scheduling approach that minimizes the makespan. The makespan is defined as the time between the start of the first scheduled task and the completion of all scheduled tasks. Load balancing is a technique to distribute incoming tasks across processing units in a way that the resource utilization is optimized and the makespan is minimized. Robotic network cloud systems can be conceptualized as graphs, with nodes representing hardware with independent computing power and edges representing data transmissions between the nodes. The initial scheduler assigns a set of newly arrived tasks to the processing units capable of performing them. To reduce the response time we can replicate some of the tasks and assign them to different processing units. This results in some tasks becoming redundant. Assigning redundant tasks refers to determining which processing unit should receive the replicated tasks. Load balancing for redundant allocation can be viewed as assigning tasks to multiple processing units with different resource sizes so that the load is evenly distributed among the units.

[†]At the date of submission Saeid Alirezazadeh was with C4-Cloud Computing Competence Center, Universidade da Beira Interior, C4-Estrada Municipal, 506, 6200-284, Covilhã, Portugal. He is now with Physikalische und Theoretische Chemie, Universität Graz, Heinrichstraße 28, Graz, 8010, Austria.

We propose a technique for load balancing, the ordered balancing algorithm (OBA), to minimize the makespan in the redundant allocation and scheduling problem. We prove theoretically the correctness of the proposed algorithm and illustrate with simulations, using R version 4.0.3, the obtained results that outperform other recent load balancing proposals.

Keywords: Cloud, Fog, Edge, Load balancing, Makespan, Redundant task scheduling, Robotic network

1 Introduction

Robotic systems are utilized in a variety of human activities, including domestic [1, 2], industrial and manufacturing [3, 4], military [5, 6], and others [7]. Various tasks, such as transporting large objects and monitoring a large area, often exceed the capabilities of a single robot. To overcome this limitation, instead of using a single robot, it is a good idea to use multiple communicating robots to perform the task cooperatively, in which case the system is called a robotic network. A robotic network's capacity is higher than that of a single robot, but it is limited by the combined capacity of all robots [8]. To increase the capacity, one can increase the number of robots, which increases the complexity of the model. Another restriction is that most tasks that involve human-robot interaction, such as object recognition [9], face recognition [10], and speech recognition [11], are computationally intensive.

Cloud robotics uses the Internet and cloud infrastructure to assign computations and also share big data in real-time [12]. Cloud robotics can help robots overcome some of their computational limitations. Deciding whether to transfer a newly received task to the cloud or process it on a server (Fog computing [13]) or execute it on one of the robots (Edge computing [14]) is a crucial feature of a cloud robotic system, called the allocation problem. The result of solving the allocation problem is a set of tasks that each processing unit should complete. The scheduling problem is concerned with arranging (scheduling) of a set of tasks given their priority, time constraints, and the order of precedence among the tasks, and answers the question of which of the assigned tasks should be completed first for each processing unit. Since tasks that do not satisfy the time window constraints and tasks' priorities cannot be executed by any processing unit, we can assume without loss of generality that the scheduling order of tasks is the order of arrival. We focus on the theoretical solution of the allocation and scheduling problem for a robotic network cloud system. We can restrict the method to the tasks to be executed by the nodes in the cloud. This is the answer to the allocation and scheduling problem that minimizes the makespan and balances the loads in the cloud infrastructure. Minimizing the makespan allows a sequence of tasks to be completed in the fastest way, and load balancing allows all computational resources to be used simultaneously without leaving any computational resource unused.

In a robotic network cloud system, each task is assigned to a processing unit for execution. The processing units may have different resources, which means that they may not be able to execute all tasks. Also, each task may require a certain amount of resources to ensure its execution. When a new set of tasks arrives in the system, the initial scheduler assigns the tasks to the processing units that are capable of performing them. We can replicate the tasks as they arrive and assign them to different processing units that can perform them to reduce response time. This results in some tasks becoming redundant. Assigning redundant tasks refers to determining which processing unit should receive the replicated tasks. New tasks arrive in the system dynamically and are assigned to processing units. Depending on where the tasks are assigned, the makespan may change. It is important to find the optimal way to schedule the newly arrived tasks so that the makespan is minimized, i.e., all processing units complete their tasks in the shortest possible time, while ensuring that a smaller number of processing units are in the idle state while the system executes all tasks. This is referred to as load balancing. Load balancing ensures that the system operates at its maximum capacity and that the available resources are used optimally.

In this paper, we propose a new method, ordered balancing, that solves the problem of load balancing in redundant task allocation in a robotic network cloud system. Our proposed method is an extension of another method, [15], but the extension is non-trivial. It is equivalent to the graph coloring problem, where nodes can only be colored from a given set of colors. This is a more complex problem that cannot be handled by existing methods, and the complexity of such problems is at least NP-complete. To solve this problem, we defined a metric that satisfies several conditions for the tasks and the appropriate processing units to which they can be assigned. Then, depending on the criterion, we call the existing result to show how each of the tasks satisfying that criterion can be assigned to different nodes, such that the makespan is minimized and the loads are balanced, see Figure 1.

We prove that our task allocation algorithm minimizes the makespan and balances the loads within all suitable nodes, and is optimal. The proof includes the precedence order between tasks. We have shown that our method is deterministic and does not involve stochastic or statistical approaches, and we have theoretically proved that it is optimal in the space of deterministic approaches to minimize makespan and balance the loads, and this has been illustrated with simulations. Our method has been rigorously tested and proven effective in analyzing large-scale cloud models. Our method enables better optimization with higher performance and maximizes efficiency.

The possibility of improving task scheduling with our method can have an impact on the efficiency of the implementation of robotic network cloud systems, since, as we show, our proposal is able to improve the state-of-the-art in this area. In addition, minimizing the makespan and balancing the loads means faster response time and better resource utilization, which can increase the profit and reduce the costs in production.

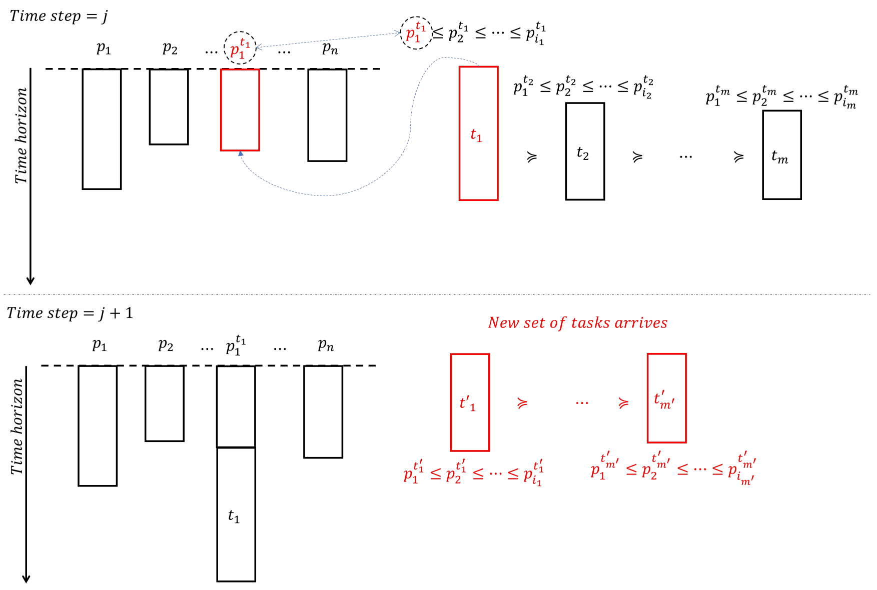
Time step = j
p_1 p_2 ... $p_1^{t_1}$ ... p_n
Time horizon
$p_1^{t_1} \leq p_2^{t_1} \leq \cdots \leq p_{i_1}^{t_1}$
$p_1^{t_2} \leq p_2^{t_2} \leq \cdots \leq p_{i_2}^{t_2}$
$p_1^{t_m} \leq p_2^{t_m} \leq \cdots \leq p_{i_m}^{t_m}$
$t_1 \succcurlyeq t_2 \succcurlyeq \cdots \succcurlyeq t_m$
Time step = j + 1
p_1 p_2 ... $p_1^{t_1}$ ... p_n
Time horizon
t_1
New set of tasks arrives
$t'_1 \succcurlyeq \cdots \succcurlyeq t'_{m'}$
$p_1^{t'_1} \leq p_2^{t'_1} \leq \cdots \leq p_{i_1}^{t'_1}$
$p_1^{t'_{m'}} \leq p_2^{t'_{m'}} \leq \cdots \leq p_{i_{m'}}^{t'_{m'}}$

Fig. 1 Overview of the ordered balancing algorithm (OBA). At time step j, the available tasks and the processing units that can perform them are sorted. Then the first task is assigned to the first processing unit in the next time step, and when a new set of tasks arrives, the union of all unscheduled tasks is sorted.

The following is the outline of the paper.

- Section 2 reviews some related work on the problem of task allocation on robotic network cloud systems.
- Section 3 describes some basic but important concepts for this work.
- Section 4 presents the key algorithm of this work.
- In Section 5, we provide a proof of the optimality and correctness of the key algorithm.
- In Section 6, we describe the experimental methodology and discuss the results.
- In Section 7, we analyze the complexity of our key algorithm compared to the state-of-the-art algorithms.
- Section 8 contains the scalability analysis of our key algorithm and the state-of-the-art algorithms.
- Finally, in Section 9, we draw some conclusions and point out future lines of work.

2 Related Work

We denote by $\mathcal{T}$ the set of all tasks that can be performed by a given multi-robot system. Suppose that $T_i \subseteq \mathcal{T}$ is a set of tasks that are newly fed to the system at the i-th step within the time horizon. As we can see in Figure 2, there are two types of task allocation:

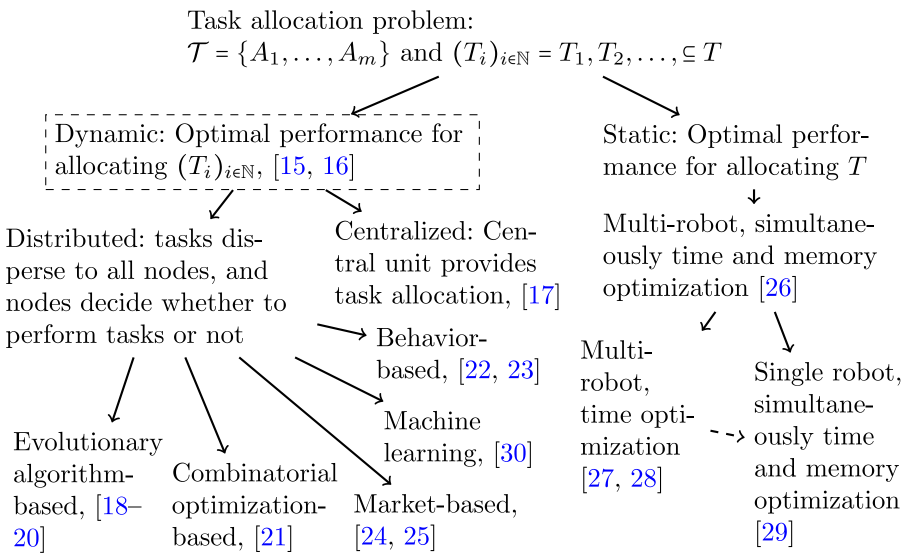
Task allocation problem:
$\mathcal{T} = \{A_1, \ldots, A_m\}$ and $(T_i)_{i\in\mathbb{N}} = T_1, T_2, \ldots, \subseteq T$
Dynamic: Optimal performance for allocating $(T_i)_{i\in\mathbb{N}}$, [15, 16]
Static: Optimal performance for allocating T
Distributed: tasks disperse to all nodes, and nodes decide whether to perform tasks or not
Centralized: Central unit provides task allocation, [17]
Multi-robot, simultaneously time and memory optimization [26]
Behavior-based, [22, 23]
Machine learning, [30]
Evolutionary algorithm-based, [18–20]
Combinatorial optimization-based, [21]
Market-based, [24, 25]
Multi-robot, time optimization [27, 28]
Single robot, simultaneously time and memory optimization [29]

Fig. 2 Diagram of studies on the subject of task allocation. Algorithm i is represented by A_i. The dashed arrow indicates that the result in [29] for a single robot cannot be extracted from the result in [27]. The dashed rectangle indicates that our proposed method belongs to the dynamic work allocation category.

- **static task allocation** seeks the optimal performance of the system in performing each task by allocating the set of all tasks;
- **dynamic task allocation** seeks the optimal performance along the time horizon by dynamically allocating the newly arrived tasks.

In static task allocation, [28] proposed an algorithm for static task allocation for a multi-robot system, but it does not consider the cloud infrastructure and communication times; [29] solved the static allocation problem by simultaneously minimizing the memory and time for a single robot cloud system, and used the concept of algebra of memory and time to deal with the precedence order between algorithms; [27] only approaches the allocation problem by considering the minimum time, and ignores the memory parameter; [26] investigated the static allocation problem for robotic network cloud systems to minimize memory and time by translating it into an optimization problem.

Dynamic task allocation can be achieved using two types of methods:

- **centralized methods**, where a centralized unit has information about the entire environment and performs task allocation, [17];
- **distributed methods**, where all tasks are distributed among all units, and they decide which tasks to perform.

Five main approaches are considered for distributed allocation:

- **evolutionary algorithm-based**, where the decision of each unit to consider for a task is made by using evolutionary operators on the space of solutions, [18, 19];
- **combinatorial optimization-based**, in which the decision of each unit to consider a task or not is made by solving the combinatorial optimization generated by the task allocation problems using an appropriate existing technique, [21];
- **market-based**, where the decision of each unit to consider or not a task is made based on an auction-based mechanism, [24];
- **machine learning-based**, where a machine learning algorithm is used to find an optimal task allocation, [30].
- **behavior-based**, where the decision of each unit to assign a task to is made based on the problem characteristics, [22];

Most studies simply transfer computationally intensive tasks to the cloud without taking into account the amount of time required for communication between robots and cloud infrastructure in task allocation in a cloud robotic system, [31–33].

Dynamic task allocation has been widely studied. The most recent works are: [23], which proposed an algorithm to improve latency, energy consumption, and computational cost for cloud robotics, taking into account the architecture of the cloud robotic, the characteristics of the task, and quality of service issues; [34] offered a centralized dynamic task allocation method by translating the problem into a non-deterministic finite automaton, and presented a lowest cost solution based on the robots' energy levels.; [35] suggested a method to deal with the tasks' deadline while minimizing the total cost; [36] examined task allocation considering that the tasks and the number of robots can change; [21] developed an optimization model based on set theoretical parameters that provides the optimal task allocation for the execution of some complex tasks by two collaborative robot teams; and [16] translates optimal task assignment into determining the largest volume of a hyperspace subspace (they studied the compatibility of a node to perform a task, communication and communication instability, and the capability of fog, cloud, and robots). Most studies on dynamic task allocation find a solution for optimal and sequential allocation of tasks to units and ignore the effects of assigned tasks on the allocation of newly arriving tasks, which may cause delays in the execution of new tasks; [30] proposed a scheduling strategy in a cloud infrastructure to reduce response time and increase resource efficiency. Reinforcement learning is used in the scheduling method. To use Bayes' theorem, the size of the occupied buffer and the total length of virtual machine tasks at each time step are assumed to be independent and the Q-values are estimated; [37] reviewed most recent papers published on scheduling in cloud computing.

For load balancing, [38] reviewed load balancing techniques on fog server; [39] proposed a centralized resource-aware load balancing model (REAL) for a batch of independent tasks considering communication time; A recent study, [15], proposed a method called Allocation of Sorted Tasks to Processing Units

(ASTPU), which describes an algorithm as a load balancing method such that incoming tasks are sorted in decreasing order by their average completion time and tasks are assigned to the processing unit with the shortest delay. They proved that their proposed method is an optimal scheduling method for load balancing. In their study, all units are identical and it is assumed that all tasks can be performed by all units, which solves a particular problem. In more realistic scenarios, different tasks usually require different resources, and we can have an architecture with different units' capabilities. In this case, each task can only be assigned to some specific units that have enough resources to execute the task, and the method proposed in [15] cannot handle this. In this paper, our main goal is load balancing considering the incoming tasks that require different resources; [40] proposed load balancing (min-min heuristic) method in Large-Scale Computing Systems (LSCSs) to jointly balance the loads of processing units and minimize the power consumption.

[41] studied load balancing as a mathematical model and proposed two greedy strategy based solutions, Basic Load Balancing Algorithm (BLBA) and Improved Load Balancing Algorithm (ILBA), where both methods measure the loads considering the precedence order of tasks and then minimize the mean square error of the loads using an optimization model. In the paper [42], the Balancer Gene Algorithm (BGA) was proposed, which assigns tasks to the virtual machines using a genetic algorithm and then uses the balancer operator to distribute the workload among the virtual machines.

Most of the researches, such as [43, 44] use random sampling, and studies like [41, 45–47] have a greedy tendency when it comes to decreasing the makespan and load balancing. For this reason, we will make a comparison using the following methods:

- **Random algorithm:** Tasks are randomly assigned to processing units, [44].
- **FIFO:** Tasks are assigned to processing units based on their arrival order and are assigned to the corresponding processing units with the smallest number of assigned tasks, [48].
- **Genetic:** Tasks are assigned to processing units using a genetic algorithm that then employs the balancer operator to balance the workload between processing units, called the Balancer Genetic Algorithm (BGA), [42].
- **Greedy:** Tasks are assigned to the processing units considering the length of their queues, [41]. We used the greedy method (ILBA), where each task is assigned to a processing unit that takes the least total time to complete all of its scheduled tasks.

Table 1 shows the advantages and limitations of each method.

As can be seen in Figure 2, the work in this paper focuses on the dynamic allocation problem. Our major goal is to propose a mechanism for allocating and scheduling newly received tasks to various computational units in such a way that the total makespan is kept to a minimum and the scheduled task completion times for all nodes are near to each other. We assume that each

Table 1 Table with advantages and limitations of each compared method.

Method	Advantage	Limitation
Random [44]	simple implementation, easy to scale, and cost efficient	uneven load distribution, no traffic prioritization, limited control, and inefficient resource utilization
FIFO [48]	fiarness, simplicity, cost efficient, and system efficiency	inefficiency for systems with varying capacity, limited flexibility, and limited scalability
BGA [42]	flexibile, efficient, and adapting to change of workload	high complexity, solution depends on the optimization parameters, not suitable for large scale scenarios
ILBA [41]	simplicity, fast and efficient, flexibible and scalable when tasks are distributed to multiple servers, and easy to modify	result is suboptimal, high computational cost, lack of flexibility to adapt new scenarios.

Table 2 Table of notation.

Notation	Description
$\preceq$	Priority order defined between tasks
P	A single processing unit
$\mathbf{P}$	The set of all processing units
T	A single task
$\mathbf{T}$	The set of newly arrived tasks
$\mathcal{T}$	The set of all tasks
Γ_T	The set of processing units capable of performing the task T

task requires a certain amount of resources for its execution and that each unit is capable of providing a certain amount of resources. The method should be independent of the optimization parameters, suitable for large scale scenarios, adaptable to new scenarios, and provide an optimal result that overcomes some of the limitations of BGA and ILBA in Table 1. We begin with a description of the mathematical tools used throughout the paper. Table 2 lists the notation used in the paper.

3 Preliminaries

Before describing the method, we explain some preliminaries related to graph theory. See [49] for more details.

Throughout the paper for a set A, we donote by $\#A$ is the number of the elements of the set A.

Definition 1 Let $G = (V, E, W)$ be a weighted graph. We say that G is a bipartite graph if there exist nonempty and disjoint subsets $\varnothing \neq V_1, V_2 \subset V$, $V_1 \cap V_2 = \varnothing$ and $V_1 \cup V_2 = V$ such that.

$$E \subseteq \{\{v_1, v_2\} \mid v_1 \in V_1, v_2 \in V_2\}. \tag{1}$$

The sets V_1 and V_2 are called parts. And the vertices of the graph is weighted with real values, $W(v_i) \in \mathbb{R}$, for $i = 1, \ldots, \#V$.

Definition 2 In a weighted bipartite graph G, a matching is a set of edges of G such that no two edges share a vertex. A maximal matching is a matching with a maximal number of edges. The size of a matching is equal to the number of edges in the matching.

We can represent a weighted bipartite graph

$$G = (V, E, W) \tag{2}$$

as

$$G = \{(V_1, \Gamma_{V_1}, W)\}, \tag{3}$$

where V_1 and V_2 are parts of G and Γ_{V_1} is a set of subsets of V_2, such that for all $v \in V_1$, $\Gamma_v \in \Gamma_{V_1}$ if and only if Γ_v is the set of vertices adjacent to v in V_2.

On a weighted bipartite graph G we define an order, $\preceq$, on Γ_{v_i}'s as follows: for $v_i, v_j \in V$:

$$\Gamma_{v_i} \preceq \Gamma_{v_j} \tag{4}$$

if one of the following occurs:

1. $\#\Gamma_{v_i} < \#\Gamma_{v_j}$;
2. $\Gamma_{v_i} = \Gamma_{v_j}$ and $W(v_i) \geq W(v_j)$;
3. $\#\Gamma_{v_i} = \#\Gamma_{v_j}$ but $\Gamma_{v_i} \neq \Gamma_{v_j}$, find in this case

$$\Delta_{v_i} = \{v_k \in A \mid \Gamma_{v_i} = \Gamma_{v_k}\} \tag{5}$$

and

$$\Delta_{v_j} = \{v_k \in A \mid \Gamma_{v_j} = \Gamma_{v_k}\}, \tag{6}$$

then $\Gamma_{v_i} \preceq \Gamma_{v_j}$ if:
(a) $\#\Delta_{v_i} \leq \#\Delta_{v_j}$;
(b) $\#\Delta_{v_i} = \#\Delta_{v_j}$ and $\max(W(\Delta_{v_j})) \geq \max(W(\Delta_{v_i}))$.

Under this new metric, $\preceq$, which is defined over the tasks in terms of occurrence relation and execution time as weights of tasks, each task is assigned to a node that is suitable for its execution with the shortest time delay. This is done assuming that the selected node is the one that has the smallest number of occurrences in the set of all suitable nodes for all other tasks that have the same number of suitable nodes to assign.

Let $\mathcal{T}$ be the set of all possible tasks that can be performed by the system. The main load balancing problem can be formulated as follows:

$$\begin{array}{rl}
\min & \text{Makespan}(\mathbf{T}, \mathbf{P}, \Gamma(\mathbf{T})) = \text{time}_f(\mathbf{T}, \mathbf{P}, \Gamma(\mathbf{T})) - \text{time}_s(\mathbf{T}, \mathbf{P}, \Gamma(\mathbf{T})) \\
\hline
s.t. & \text{time}_s(\mathbf{T}, \mathbf{P}, \Gamma(\mathbf{T})) = \min\{\text{Startprocessing}(P, T) \mid T \in \mathbf{T}, P \in \Gamma(\mathbf{T}))\} \\
& \text{time}_f(\mathbf{T}, \mathbf{P}, \Gamma(\mathbf{T})) = \max\{\text{Completeprocessing}(P, T) \mid T \in \mathbf{T}, P \in \Gamma(\mathbf{T}))\} \\
& \mathbf{T} \subseteq \mathcal{T} \text{ is the set of newly arrived tasks,} \\
& \mathbf{P} \text{ is the set of all processing units,} \\
& \Gamma(\mathbf{T}) = \{\Gamma_T \mid T \in \mathbf{T}\},
\end{array}$$

$\Gamma_T = \{P \in \mathbf{P} \mid P \text{ can perform } T\}$,

Startprocessing(P, T) is the time to start processing task T by processing unit P,

Completeprocessing(P, T) is the time to complete processing task T by processing unit P,

$\#\Gamma_T > 0$,

all tasks in $\mathcal{T}$ have time window constraints.

The presence of time window constraints is handled by the grid of all tasks, [15].

4 Key Method

We first describe the ordered balancing algorithm (OBA) that plays the key role in the paper. OBA sorts the set of newly arrived tasks and the set of processing units and allocates each task to the most suitable processing unit based on the obtained order, as detailed in Algorithm 1. The proof of the correctness and optimality of the algorithm is due to the existence of a proof of Proposition 1.

Algorithm 1 Pseudocode of ordered balancing algorithm (OBA)

```
Input: $\mathbf{T}$ (the set of all newly arrived tasks), $\mathbf{P}$ (the set of scheduled tasks
    for all processing units), and $\Gamma(T)$ (the set of processing units that can
    perform each of the newly arrived tasks).
Output: $M : \mathbf{T} \to \mathbf{P}$ (the mapping from the set of newly arrived tasks to the
    set of processing units).
1: while $\mathbf{T} \neq \varnothing$ do    ▷ There is at least one task to be assigned to a processor.
2:     Sort the sets $\mathbf{T}$ in decreasing order    ▷ Considering the metric $\preceq$.
3:     Sort the set $\mathbf{P}$ in increasing order    ▷ Considering the expected completion time of all their scheduled tasks.
4:     $M(T) = P$.    ▷ Assign the first task $T$ of $\mathbf{T}$ (the sorted set) to the first processing unit $P$ of $\mathbf{P}$ (the sorted set) that is capable of performing the task $T$.
5:     $\mathbf{T} = \mathbf{T} \setminus \{T\}$.    ▷ Update the set of tasks.
6:     $P = P \cup \{T\}$.    ▷ Update the set of scheduled tasks for the processing units. The set of scheduled tasks for all processing units remains unchanged, except for processing unit $P$, which is assigned task $T$ and added to its scheduled tasks.
7: end while
8: return The mapping $M$.
```

The detailed description of Algorithm 1 is as follows: The OBA algorithm takes the set of newly arrived tasks

$$\mathbf{T} = \{T_1, \dots, T_n\}, \tag{7}$$

the set of scheduled tasks for processing units

$$\mathbf{P} = \{P_1, \dots, P_m\}, \tag{8}$$

and

$$\Gamma(T) = \{\Gamma_{T_1}, \dots, \Gamma_{T_n}\} \tag{9}$$

the set of processing units that can perform each of the newly arrived tasks, as the input. And produces the mapping

$$M : \mathbf{T} \to \mathbf{P} \tag{10}$$

from the set of newly arrived tasks to the set of processing units, as the output. The OBA algorithm then recursively applies the following steps until the set is $\mathbf{T} = \varnothing$:

1. Sort the set $\mathbf{T}$ in decreasing order considering $\preceq$ and sort the set $\mathbf{P}$ in increasing order considering the expected completion time of scheduled tasks of all processing units.
2. Take out the first task T from $\mathbf{T}$ (ordered set as defined in 1) and assign it to the first processing unit $P \in \Gamma(T)$ from $\mathbf{P} \cap \Gamma(T)$ (ordered set as defined in 1), and let $M(T) = P$.
3. Update the set of tasks $\mathbf{T} = \mathbf{T} \setminus \{T\}$.
4. Update the processing units' set of scheduled tasks. The set of scheduled tasks for all processing units remains unchanged, except for processing unit P, which is assigned task T and added to its scheduled tasks.

When the iteration completes, the mapping M defined in 2 for each task will be obtained. The flowchart of the algorithm 1 is shown in Figure 3.

This method is an extension and more generalized version of the ASTPU method in [15]. The paper [15] assumes that all processing units are capable of performing tasks. However, in this paper we assume that tasks can only be performed by some specific processing units, not all. Such an extension is not trivial, and to find an optimal task scheduling, we need to consider the capabilities of the processing units, the length of the scheduled tasks, and the execution time of the newly arrived tasks. We combine these three orders into a single order (4) and then prove in Proposition 1 that optimal scheduling with minimal time is possible with this newly defined order. The grid of all tasks is developed and designed in [15], which allows us to handle the precedence order between tasks by letting each task take its specific position in the grid and then assigning the stream of the grid to different processing units.

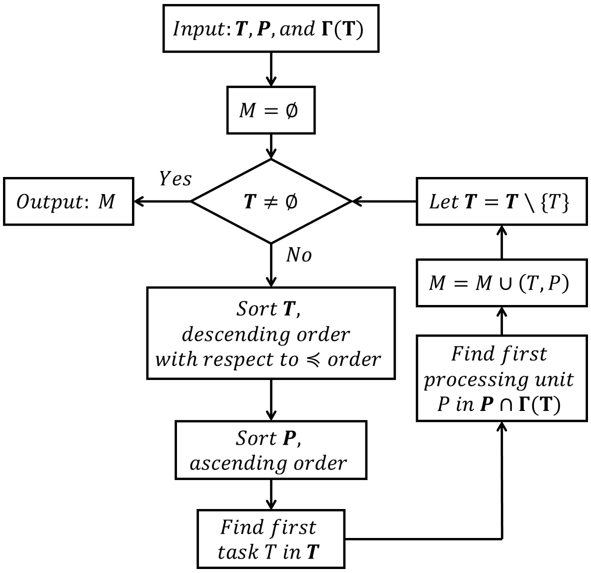
Input: **T**, **P**, and Γ(**T**)
M = ∅
T ≠ ∅
Yes
Output: M
No
Sort **T**, descending order with respect to ≼ order
Sort **P**, ascending order
Find first task T in **T**
Find first processing unit P in **P** ∩ Γ(**T**)
M = M ∪ (T, P)
Let **T** = **T** \ {T}

Fig. 3 Flowchart of the ordered balancing algorithm (OBA).

5 Proof of Optimality

The following proposition shows that the ordered balancing algorithm 1 is an optimal task allocation method in the sense that all processing units can finish their tasks practically simultaneously. We show that assigning tasks to processors in a different way than the ordered balancing algorithm 1 increases the variance, meaning that the loads will be less balanced.

Proposition 1 *(OBA method) Let*

$$A = \{a_i \mid i = 1, \dots, n\} \tag{11}$$

be a list of positive real numbers, $m \geq 1$ be an integer, and

$$\Gamma_{a_i} \subseteq \{1, \dots, m\}, \tag{12}$$

with Γ_{a_i} be the set of places where a_i can be located. Consider randomly removing of one a_i's and putting it in exactly one of the places in Γ_{a_i} and repeating the same process until the set A is empty. Define random variables X_j for $j = 1, \dots, m$ to be the sum of a_i's at place j. Then, the ordered balancing algorithm 1 minimizes the variance of X_i's.

Moreover, when $A = \{a_1, \dots, a_n\}$, with $m \geq n$, and Γ_{a_i}'s describe a maximal matching of A to $\{1, \dots, m\}$, one can obtain the minimal variance by placing each of the a_i's at its matching location on the set $\{1, \dots, m\}$.

Proof We first show that if $A = a_1, \dots, a_n$, $m \geq n$, and Γ_{a_i} describes a maximal matching of A with $\{1, \dots, m\}$ of size $\#A$, the minimal variance can be obtained by putting each a_i at the location of its match on the set $\{1, \dots, m\}$. Then, the result of having a minimum variance is given by the ASTPU proposition in [15].

For the optimal allocation with minimal variance, we consider the order, $\preceq$ defined in (4), on Γ_{a_i}'s. Without loss of generality, we can assume that

$$\Gamma_{a_1} \preceq \Gamma_{a_2} \preceq \ldots \preceq \Gamma_{a_n}. \tag{13}$$

Now we apply the ASTPU proposition [15] for the allocation a_i's, with respect to the order $\preceq$, to the particular place $j \in \Gamma_{a_i}$ with the smallest value X_j. If there are more than one suitable places with the smallest value, then put a_i into the one that has the smallest number of occurrences in all $\Gamma(a_k)$, with $\#\Gamma_{a_i} = \#\Gamma_{a_k}$. From the ASTPU proposition, [15], setting each a_i with this method gives the smallest variance with respect to its appropriate places.

Note that the defined order $\preceq$ prioritizes the a_i's. The idea for placing the elements of A is first to place the elements that have the smallest number of suitable places because other elements of A, with a larger number of suitable places, have more opportunities to be placed. Given the smaller sum of a_i's in the positions where the elements of A are placed, and the ASTPU proposition, [15], it implies that this placement produces a smaller variance increase. That is, the condition 1 implies that the element a_i should be placed first before the placement of a_j if $\#\Gamma_{a_i} < \#\Gamma_{a_j}$. In this case, a_i has a smaller number of suitable positions to be placed in.

Condition 2 implies that if two elements a_i and a_j have the same set of suitable places, then the one with the larger value should be placed first, which is the equivalent way of saying that the ASTPU proposition [15] should only be applied over the fixed set of suitable places.

Conditions 3 and 33a, imply that if two elements a_i and a_j have the same number of suitable places, then, the one that has the smaller number of identical sets of suitable places should be placed first. This is because, in this case, we are adding a smaller number of elements in certain places.

The conditions 3 and 33b, imply that if two elements a_i and a_j have the same number of matching positions and have the same number of identical sets of matching positions, then the one that has the same set of matching positions of a larger value should be placed first. This implies that the largest values should be placed first, and the result is given by the ASTPU proposition [15]. $\square$

6 Experiments

The experiments were performed on a HP Laptop 15-dw2xxx with Intel Core i5 10th generation with processor Intel(R) Core(TM) i5-1035G1 CPU @ 1.19 GHz, RAM 16.0 GB, x64-based processor, and we used R version 4.0.3. We compared our result with random algorithm [44], FIFO [48], Balancer Genetic Algorithm (BGA) [42], and a greedy algorithm (ILBA) [41]. The code is made available at: https://github.com/SaeidZadeh/LoadBalancer.

6.1 Experiment 1

We consider 10 processing units and the total number of tasks is unknown. Each of the newly arrived tasks' average execution time is chosen at random from the range $[0, 10000]$ milliseconds. We then randomly select the set of appropriate processing units that can execute each of the tasks.

As we have shown in Proposition 1, when there are a large number of processing units, the optimal task allocation to the processing units can be done

Table 3 Average execution time (in milliseconds) of all methods applied to the last settings for scheduling tasks with 1000 time steps.

Parameter	Value
Number of Processing units	10
average execution time of each task	$[0, 10000]$ milliseconds
Task arrival rate	constant $n \in \{1, \dots, 200\}$
Number of iterations	50
Number of time steps	1

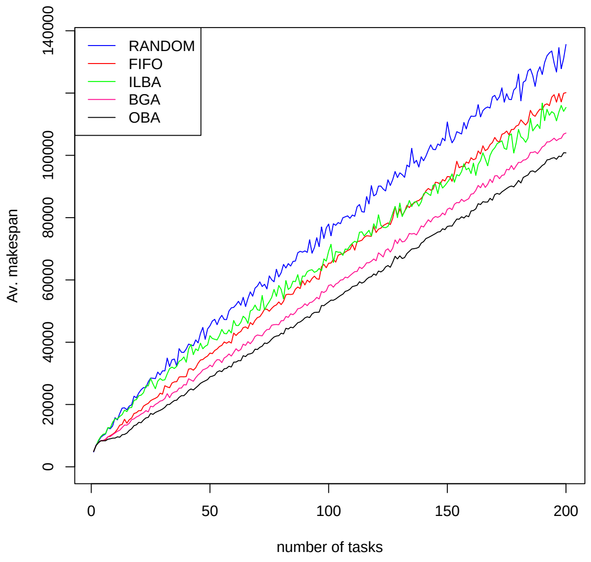

Fig. 4 Average makespans (in milliseconds) in a single time step.

by assigning at most one task to each processing unit, and in this situation the performance of the state-of-the-art methods and our proposed method is indistinguishable. To avoid such a scenario, we consider the number of processing units to be equal to 10. Moreover, the execution time of the tasks can be either slow or fast. Within the interval $[0, 10000]$ milliseconds, we consider tasks that can be executed in short and long time and that are randomly generated.

We consider that the task arrival rate is fixed (the number of tasks that arrive in all time steps is constant), and varies from 1 to 200 tasks in a single time step. For each task arrival rate $n \in \{1, \dots, 200\}$, we randomly select 50 tasks according to their execution time interval $[0, 10000]$. For all 50 repetitions of task selection, we apply all task allocation algorithms and find their average makespan considering the task arrival rate n. Figure 4 shows the average makespan of the different methods. As can be seen in Figure 4, in a single time step OBA outperforms all the other methods when the number of newly arrived tasks is larger than the number of processing units.

Table 3 shows the detailed simulation setup.

Another advantage of OBA is that all processing units complete their scheduled tasks at almost the same time. The Task Completion time Difference

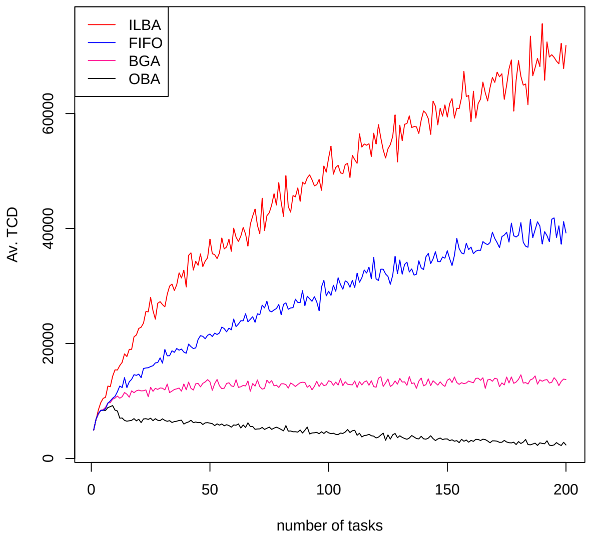

Fig. 5 TCD as a function of the number of arriving tasks in a single time step (in milliseconds). When compared to the OBA approach, it can be shown that ILBA, BGA, and FIFO have growing TCD.

(TCD) is used in [15] as a measure to check how balanced the loads of all processing units are:

$$TCD = t_{\max} - t_{\min}, \tag{14}$$

where $t_{\min}$ is the shortest time in which one of the processing units has completed all of its scheduled tasks, and $t_{\max}$ is the total time in which all processing units have completed all of their scheduled tasks. Figure 5 shows the changes of TCD as a function of the number of arrived tasks. From Figures 4 and 5, it can be seen that OBA minimizes the makespan and has better performance in balancing the loads.

6.2 Experiment 2

Now consider that the arrival rate of the tasks is a random variable from the Poisson distribution with parameter 10 within the total time steps of 200. In this setting we assume that at each time step all processing units has no scheduled tasks, i.e., before new set of tasks arrives all processing units complete their scheduled tasks. We repeat the same procedure 50 times. First, for each time step, we independently compute the average makespan, TCD, and the difference between OBA, Random method, ILBA, BGA, and the FIFO method. The average makespan with respect to each method is shown in Figure 6, where we have averaged all the average makespans of all 50 repetitions at the end. As can be seen in Figure 6, if a new set of tasks arrives after all processing units complete their scheduled tasks, then OBA better minimizes the makespan compared with all the other methods.

Table 4 shows the detailed simulation setup.

Table 4 Average execution time (in milliseconds) of all methods applied to the last settings for scheduling tasks with 1000 time steps.

Parameter	Value
Number of Processing units	10
average execution time of each task	$[0, 10000]$ milliseconds
Task arrival rate	random Poisson distribution with parameter 10
Number of iterations	50
Number of time steps	200
Time steps	at completion of all scheduled tasks

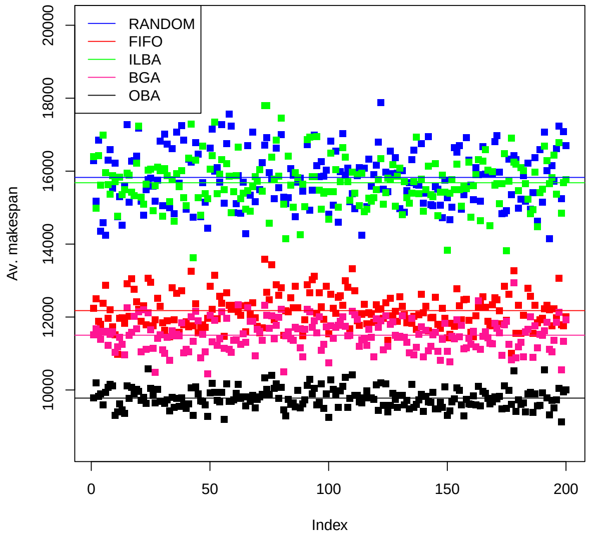

Fig. 6 Average makespans (in milliseconds) per each of the 200 time steps for 50 repetitions with a Poisson distribution with parameter 10 changing the number of arrival tasks. The average makespans of all experiments with respect to each method are represented by straight lines.

Now we take the average of the 50 repetitions' makespans and repeat the experiment 50 times more. The mean of the average makespans of the different methods for all the 50 runs are

$$15915.79(\pm 62.71),\ 12159.51(\pm 37.52),\ 15713.71(\pm 60.08),$$
$$11502.77(\pm 33.02),\ 9752.72(\pm 18.92)$$

for Random, FIFO, ILBA, BGA, and OBA, respectively.

The mean TCD's are

$$10494.06(\pm 30.65), 7852.82(\pm 18.63), 40.93(\pm 0.08)$$

for ILBA, BGA, and OBA, respectively. So, the tasks are better distributed among the processing units with OBA, and it reduces the average makespan.

Table 5 Average execution time (in milliseconds) of all methods applied to the last settings for scheduling tasks with 1000 time steps.

Parameter	Value
Number of Processing units	100
average execution time of each task	$[0, 10000]$ milliseconds
Task arrival rate	random Poisson distribution with parameter 200
Number of iterations	50
Number of time steps	1000
Time steps	any, while processing units performing tasks

Table 6 Average execution time (in milliseconds) of all methods applied to the last settings for scheduling tasks with 1000 time steps.

Method	OBA	FIFO	Random	ILBA	BGA
Average Time	260.47	44.53	80.90	142.28	146.60

6.3 Experiment 3

Now consider the total time step to be 1000, the Poisson distribution variate to be the parameter 200, and the number of processing units to be 100. We also consider that time steps are while some processing units performing their scheduled tasks, i.e, new set of tasks arrives while some processing units have some scheduled tasks to perform. We repeat the same procedure 50 times. But now we calculate the average makespan, TCD, and the differences between OBA and ILBA, BGA, and FIFO methods for all 1000 time steps.

Table 5 shows the detailed simulation setup.

The average makespan with respect to each method is shown in Figure 7, where we take the mean of all the average makespans of all 50 repetitions at the end. In Figure 7 the mean of differences between ILBA and OBA is 1993.32(±209.80) and the mean of differences between BGA and OBA is 2542.84(±301.85). The average TCD in each of 1000 time steps for all the methods does not have significant changes. The mean and standard deviation of the values for the average TCD in each of 1000 time steps for the methods ILBA, BGA, and OBA are 10105.86(±207.42), 12128.87(±303.69), 9070.40(±68.63), respectively, showing that ILBA and BGA always has a larger average TCD compared to OBA.

Our result shows that BGA performs better than both FIFO and ILBA in the short run, and ILBA performs better than both BGA and FIFO in the long run. In both cases, OBA performs better than either of them.

In this experiment, all methods were used to measure the average time to complete task scheduling after 1000 time steps. The values are listed in Table 6. First take the average time of 50 iterations, then add the value of 1000 time steps. The result in Table 6 shows that although OBA is slower than BGA and ILBA, it reduces the average makespan by at least 1500 milliseconds compared to ILBA and BGA, as the mean of differences between ILBA and OBA is 1993.32(±209.80) so $1993.32 - 2 * 209.80 > 1500$, which is a significant improvement compared to the execution time.

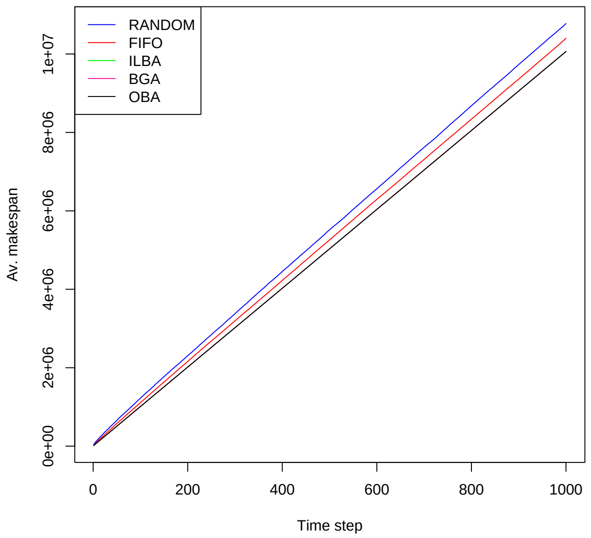

Fig. 7 Average makespans (in milliseconds), for the overall 1000 time steps, for 50 repetitions where the number of arrival tasks changes following a Poisson distribution with parameter 200. The differences between ILBA, BGA, and OBA are difficult to discern in this figure. Nevertheless, the mean of the differences between ILBA and OBA is 1993.32(±209.80) and the mean of the differences between BGA and OBA is 2542.84(±301.85).

The Violin plot for time step 1000 are used to compare the kernel densities of all the methods. For better illustration, all the average makespans are divided by their maximum value within all methods, Figure 8. In this figure, the kernel densities of the OBA, ILBA, and BGA are similar and shows that all have a similar pattern to the Gaussian distribution. Therefore, their results can be compared using the mean and standard deviation of their differences, where the mean of the differences between ILBA and OBA is 1993.32(±209.80) and the mean of the differences between BGA and OBA is 2542.84(±301.85).

To compare the difference between the data collected using different methods, we perform a t-test for time step 1000. In Table 7 we show the t-values along with the p-values in the t-test. From the first row of the Table 7, since all the p-values are smaller than 0.05 and all the t-values are larger than 1.96, OBA is significantly different from all other methods.

7 Complexity

Let m be the number of newly arrived tasks and n the number of processors. The complexity of general dynamic scheduling can be easily determined as follows: Each task should be assigned to the appropriate processor. Thus, if n_i is the number of matching processors that can execute the i-th task, then $M = \sum_{i=1}^{m} n_i$ is the total constraint on assigning tasks to different processing units. Now, to find the optimal solution, we need to find the minimum makespan within these constraints that has complexity $O(M)$, in the worst case $M = nm$,

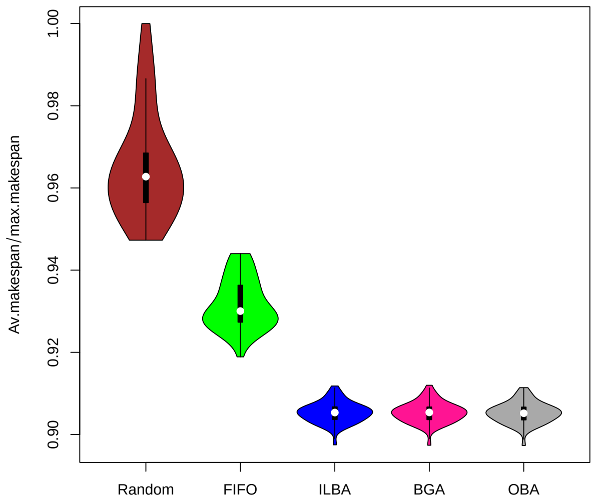

Fig. 8 Comparing the kernel densities of all methods for time step 1000, all the average makespans are divided by their maximum value within all methods.

Table 7 Comparing t-values (p-values) for the data obtained using Random, FIFO, BGA, ILBA, and OBA for time step 1000.

	OBA	FIFO	Random	ILBA	BGA
OBA	-	t = -45.00 (p < 2.2e-16)	t = -42.81 (p < 2.2e-16)	t = -3.25 (p = 0.002)	t = -7.10 (p = 4.6e-9)
FIFO	t = 45.00 (p < 2.2e-16)	-	t = -20.01 (p < 2.2e-16)	t = 45.02 (p < 2.2e-16)	t = 44.60 (p < 2.2e-16)
Random	t = 42.81 (p < 2.2e-16)	t = 20.01 (p < 2.2e-16)	-	t = 42.75 (p < 2.2e-16)	t = 42.65 (p < 2.2e-16)
ILBA	t = 3.25 (p = 0.002)	t = -45.02 (p < 2.2e-16)	t = -42.75 (p < 2.2e-16)	-	t = -3.84 (p = 0.0004)
BGA	t = 7.10 (p = 4.6e-9)	t = -44.60 (p < 2.2e-16)	t = -42.65 (p < 2.2e-16)	t = 3.84 (p = 0.0004)	-

and the complexity is $O(nm)$. In OBA, the idea is to sort the tasks and processing units and then assign the tasks to the respective processor based on the obtained order. The complexity of OBA is then

$$O(\max\{m\log(m), n_i\log(n_i)\}), \tag{15}$$

and in the worst case, the complexity is

$$O(\max\{m\log(m), n\log(n)\}). \tag{16}$$

The complexity of the genetic algorithm is $O(\max\{gn_im\})$, where g is the number of generations, so in the worst case the complexity is $O(gnm)$. And the complexity of ILBA, which is only about finding the processing unit with minimum execution time, is $O(\max\{n_i\})$, in the worst case the complexity is $O(n)$.

8 Scalability

To better illustrate the scalability in two dimensional space we consider two settings.

- **Setting 1** fix the number of processing units equal to 256, change the average number of arrival tasks as $2, 4, \dots, 8192$, and measure the average execution time (in milliseconds) of methods ILBA, BGA, and OBA as functions of average number of arrival tasks, Figure 9;
- **Setting 2** fix the average number of tasks equal to 256, change the number of processing units as $2, 4, \dots, 8192$, and measure the average execution time (in milliseconds) of methods ILBA, BGA, and OBA as functions of number of processing units, Figure 10.

For both settings we consider only the total time steps of 20 and each experiment is repeated 50 times. Figures 11 and 12 show the minimum improvement of the average makespan of BGA and ILBA compared with OBA as a function of average number of tasks and number of processing units for setting 1 and setting 2, respectively. Figures 9 and 11 show that the growth rates of the average execution time of ILBA, BGA, and OBA are similar and differ only by constant factors. Figure 10 shows that for a fixed number of processing units, increasing the number of tasks further improves the average execution time of OBA compared to BGA and ILBA. Figure 12 shows that for a fixed number of arrival tasks, increasing the number of processing units further improves the average makespan obtained by OBA compared to BGA and ILBA for a small number of processing units. When the number of processing units is very large (much larger than the number of arrival tasks), the Pigeonhole principle suggests that each processing unit should be assigned at most one task to minimize the makespan so that all methods have the same makespan.

9 Conclusion and Future Work

The execution of each task requires a certain amount of resources, and we have assumed that each unit can only provide a limited amount of resources. We have proposed a method to allocate the newly arrived tasks to different processing units such that the scheduled load of all processing units is balanced and the minimum makespan is achieved. The OBA, Ordered Balancing Algorithm, can handle load balancing in the presence of redundant task allocation. It is a non-trivial extension of the ASTPU method [15]. We compared OBA with existing scheduling methods. We proved the correctness of OBA and

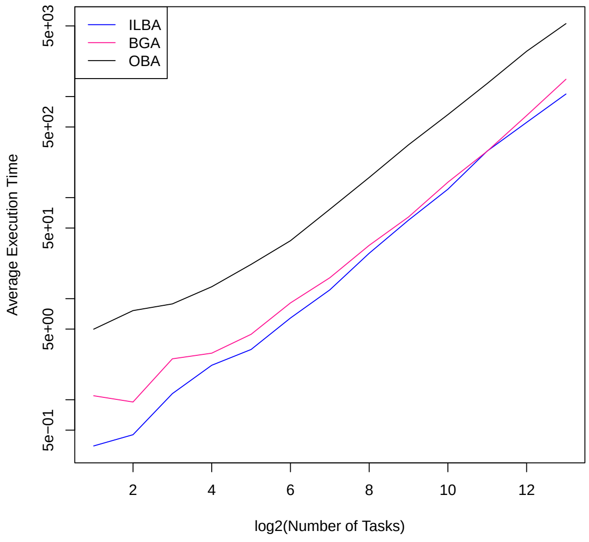

Fig. 9 The average execution time (in milliseconds) of methods ILBA (in blue), BGA (in red), and OBA (in black) in setting 1 as functions of average number of arrival tasks when the number of processing units is 256 and the average number of newly arriving task is $2, 4, \ldots, 8192$. x axis is in logarithmic, $\log_2$, scale.

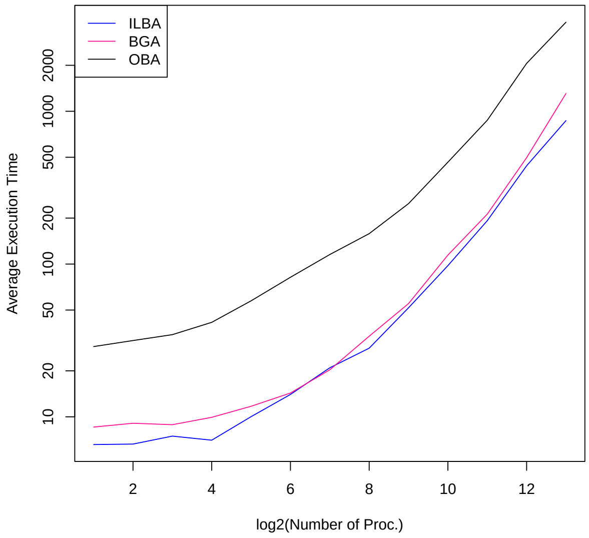

Fig. 10 The average execution time (in milliseconds) of methods ILBA (in blue), BGA (in red), and OBA (in black) in setting 2 as functions of number of processing units when the number of processing units is $2, 4, \ldots, 8192$ and the average number of newly arriving task is 256. x axis is in logarithmic, $\log_2$, scale.

illustrated through simulations its performance compared to four other methods and showed that it outperforms all of them. Our result shows that when

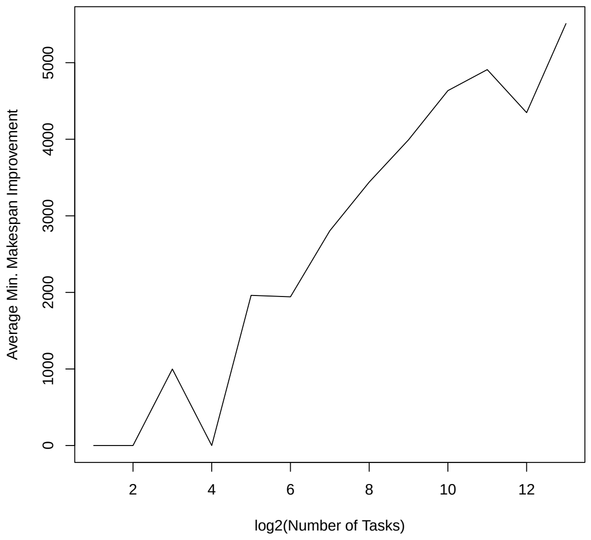

Fig. 11 The minimum improvement of the average makespan (in milliseconds) of BGA and ILBA compared with OBA in setting 1 as a function of average number of arrival tasks. x axis is in logarithmic, $\log_2$, scale.

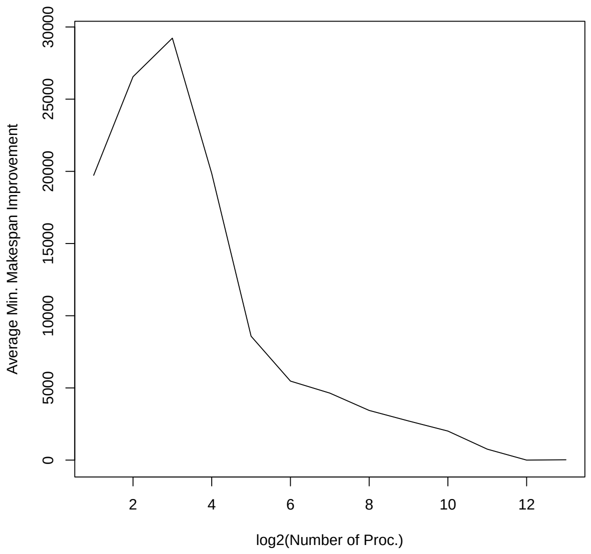

Fig. 12 The minimum improvement of the average makespan (in milliseconds) of BGA and ILBA compared with OBA in setting 2 as a function of number of processing units. x axis is in logarithmic, $\log_2$, scale.

the average number of newly arrived tasks is larger than the number of processing units, OBA has better performance for load balancing and minimizing makespan.

OBA is a method that can be added to any processing for distributed task assignment, or it can be added to the centralized entity. In the distributed case, the task arrives at a processing unit and all processing units exchange information about their current scheduled length and the length of new tasks that have arrived and are to be assigned. Then the task is migrated to an appropriate processing unit after OBA is performed. In the case of a centralized entity, the length of the tasks to be assigned along with the length of the scheduled tasks to each processing unit is communicated to the central unit, then OBA is applied and the tasks are migrated to the appropriate processing units determined by OBA.

For future work, we will include communication instabilities [16] and possible processing unit failures, for more realistic scenarios.

Acknowledgments. This work was supported by operation Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, cofinanced by the European Regional Development Fund (ERDF) through the Programa Operacional Regional do Centro (Centro 2020), in the scope of the Sistema de Apoio à Investigação Cientifíca e Tecnológica - Programas Integrados de IC&DT. This work was supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT-Fundação para a Ciência e a Tecnologia, through national funds.

Declarations

- Ethics approval: We declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere.
- Consent to participate: Not applicable.
- Consent for publication: Not applicable.
- Availability of data and materials: Not applicable.
- Code availability: The code is made available at https://github.com/SaeidZadeh/LoadBalancer.
- Competing interests: The authors declare that they have no conflict of interest.
- Fundings: Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, cofinanced by the European Regional Development Fund (ERDF) through the Programa Operacional Regional do Centro (Centro 2020). NOVA LINCS (UIDB/04516/2020) with the financial support of FCT-Fundação para a Ciência e a Tecnologia
- Authors' contributions: Saeid Alirezazadeh did conceptualization, investigation, methodology, formal analysis, software, validation, writing-original draft, and visualization. Luís Alexandre did conceptualization, funding acquisition, supervision, writing, reviewing and editing, and visualization. All authors reviewed the manuscript.
- Acknowledgments: This work was supported by operation Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing,

cofinanced by the European Regional Development Fund (ERDF) through the Programa Operacional Regional do Centro (Centro 2020), in the scope of the Sistema de Apoio à Investigação Científica e Tecnológica - Programas Integrados de IC&DT. This work was supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT-Fundação para a Ciência e a Tecnologia, through national funds.

References

[1] Prassler, E., Kosuge, K.: In: Siciliano, B., Khatib, O. (eds.) Domestic Robotics, pp. 1253–1281. Springer, Germany (2008). https://doi.org/10.1007/978-3-540-30301-5_55

[2] Xu, Y., Qian, H., Wu, X.: Household Service Robotics. Elsevier Science, China (2014)

[3] Ross, L.T., Fardo, S.W., Walach, M.F.: Industrial Robotics Fundamentals: Theory and Applications, 3rd edn. Goodheart-Willcox Co., USA (2017)

[4] International Federation of Robotics IFR worldwide, I.: International Federation of Robotics IFR, Industrial Robots. https://www.ifr.org/industrial-robots

[5] Springer, P.J.: Military Robots and Drones: A Reference Handbook. Contemporary world issues. ABC-CLIO, USA (2013)

[6] Nath, V., Levinson, S.E.: Autonomous Military Robotics. Springer, USA (2014)

[7] Siciliano, B., Khatib, O.: Springer Handbook of Robotics, 2nd edn. Springer, USA (2016)

[8] Hu, G., Tay, W.P., Wen, Y.: Cloud robotics: architecture, challenges and applications. IEEE Network **26**(3), 21–28 (2012). https://doi.org/10.1109/MNET.2012.6201212

[9] Xu, G., Zhang, Z.: Epipolar Geometry in Stereo, Motion and Object Recognition: a Unified Approach vol. 6. Springer, China (2013)

[10] Jain, A.K., Li, S.Z.: Handbook of Face Recognition vol. 1. Springer, Netherland (2011)

[11] Jelinek, F.: Statistical Methods for Speech Recognition. MIT press, Netherland (1997)

[12] Kehoe, B., Patil, S., Abbeel, P., Goldberg, K.: A survey of research on

cloud robotics and automation. IEEE Transactions on Automation Science and Engineering **12**(2), 398–409 (2015). https://doi.org/10.1109/TASE.2014.2376492

[13] Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. MCC-12, pp. 13–16. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2342509.2342513

[14] Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal **3**(5), 637–646 (2016). https://doi.org/10.1109/JIOT.2016.2579198

[15] Alirezazadeh, S., Alexandre, L.A.: Improving makespan in dynamic task scheduling for cloud robotic systems with time window constraints. Cluster Computing (2022). https://doi.org/10.1007/s10586-022-03724-x

[16] Alirezazadeh, S., Alexandre, L.A.: Dynamic task allocation for robotic network cloud systems. In: 2020 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom), pp. 1221–1228 (2020). https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00181

[17] Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. Society for Industrial and Applied Mathematics, USA (2012). https://doi.org/10.1137/1.9781611972238

[18] Tsiogkas, N., Lane, D.M.: An evolutionary algorithm for online, resource-constrained, multivehicle sensing mission planning. IEEE Robotics and Automation Letters **3**(2), 1199–1206 (2018)

[19] Arif, M.U.: An evolutionary algorithm based framework for task allocation in multi-robot teams. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. AAAI-17. AAAI Press, USA (2017)

[20] Cheng, Z., Li, P., Wang, J., Guo, S.: Just-in-time code offloading for wearable computing. IEEE Transactions on Emerging Topics in Computing **3**(1), 74–83 (2015)

[21] Wang, H., Chen, W., Wang, J.: Coupled task scheduling for heterogeneous multi-robot system of two robot types performing complex-schedule order fulfillment tasks. Robotics and Autonomous Systems, 103560 (2020). https://doi.org/10.1016/j.robot.2020.103560

[22] Parker, L.E.: Alliance: an architecture for fault tolerant multirobot cooperation. IEEE Transactions on Robotics and Automation **14**(2), 220–240 (1998)

[23] Chen, W., Yaguchi, Y., Naruse, K., Watanobe, Y., Nakamura, K.: Qos-aware robotic streaming workflow allocation in cloud robotics systems. IEEE Transactions on Services Computing, 1–14 (2018)

[24] He, J., Badreldin, M., Hussein, A., Khamis, A.: A comparative study between optimization and market-based approaches to multi-robot task allocation. Advances in Artificial Intelligence **2013**, 256524 (2013). https://doi.org/10.1155/2013/256524

[25] Wang, L., Liu, M., Meng, M.Q.: A hierarchical auction-based mechanism for real-time resource allocation in cloud robotic systems. IEEE Transactions on Cybernetics **47**(2), 473–484 (2017). https://doi.org/10.1109/TCYB.2016.2519525

[26] Alirezazadeh, S., Correia, A., Alexandre, L.A.: Optimal algorithm allocation for robotic network cloud systems. Robotics and Autonomous Systems, 104144 (2022). https://doi.org/10.1016/j.robot.2022.104144

[27] Li, S., Zheng, Z., Chen, W., Zheng, Z., Wang, J.: Latency-aware task assignment and scheduling in collaborative cloud robotic systems. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 65–72 (2018). https://doi.org/10.1109/CLOUD.2018.00016

[28] Lin, C.-F., Tsai, W.-H.: Optimal assignment of robot tasks with precedence for muliti-robot coordination by disjunctive graphs and state-space search. Journal of Robotic Systems **12**(4) (1995) https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.4620120402. https://doi.org/10.1002/rob.4620120402

[29] Alirezazadeh, S., Alexandre, L.A.: Optimal algorithm allocation for single robot cloud systems. IEEE Transactions on Cloud Computing, 1–13 (2021). https://doi.org/10.1109/TCC.2021.3093489

[30] Mostafavi, S., Hakami, V.: A stochastic approximation approach for foresighted task scheduling in cloud computing. Wireless Personal Communications (2020). https://doi.org/10.1007/s11277-020-07398-9

[31] Arumugam, R., Enti, V.R., Bingbing, L., Xiaojun, W., Baskaran, K., Kong, F.F., Kumar, A.S., Meng, K.D., Kit, G.W.: Davinci: A cloud computing framework for service robots. In: 2010 IEEE International Conference on Robotics and Automation, pp. 3084–3089 (2010)

[32] Gouveia, B.D., Portugal, D., Silva, D.C., Marques, L.: Computation

sharing in distributed robotic systems: A case study on slam. IEEE Transactions on Automation Science and Engineering **12**(2), 410–422 (2015)

[33] Hunziker, D., Gajamohan, M., Waibel, M., D'Andrea, R.: Rapyuta: The roboearth cloud engine. In: 2013 IEEE International Conference on Robotics and Automation, pp. 438–444 (2013)

[34] Schillinger, P., Bürger, M., Dimarogonas, D.V.: Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. The International Journal of Robotics Research **37**(7), 818–838 (2018) https://arxiv.org/abs/https://doi.org/10.1177/0278364918774135. https://doi.org/10.1177/0278364918774135

[35] Zhang, P., Zhou, M.: Dynamic cloud task scheduling based on a two-stage strategy. IEEE Transactions on Automation Science and Engineering **15**(2), 772–783 (2018)

[36] Chen, X., Zhang, P., Du, G., Li, F.: A distributed method for dynamic multi-robot task allocation problems with critical time constraints. Robotics and Autonomous Systems **118**, 31–46 (2019). https://doi.org/10.1016/j.robot.2019.04.012

[37] Ahmad, Z., Jehangiri, A.I., Ala'anzy, M.A., Othman, M., Latip, R., Zaman, S.K.U., Umar, A.I.: Scientific workflows management and scheduling in cloud computing: Taxonomy, prospects, and challenges. IEEE Access **9**, 53491–53508 (2021). https://doi.org/10.1109/ACCESS.2021.3070785

[38] Haghi Kashani, M., Mahdipour, E.: Load balancing algorithms in fog computing: A systematic review. IEEE Transactions on Services Computing, 1–1 (2022). https://doi.org/10.1109/TSC.2022.3174475

[39] Alam, M., Haidri, R.A., Shahid, M.: Resource-aware load balancing model for batch of tasks (bot) with best fit migration policy on heterogeneous distributed computing systems. International Journal of Pervasive Computing and Communications **16**(2), 113–141 (2020). https://doi.org/10.1108/IJPCC-10-2019-0081

[40] Sardar Khaliq uz Zaman, M.A.K.B.S.A.M.A.u.R.K. Tahir Maqsood: A load balanced task scheduling heuristic for large-scale computing systems. Computer Systems Science and Engineering **34**(2), 79–90 (2019). https://doi.org/10.32604/csse.2019.34.079

[41] Sun, Y., Mao, S., Huang, S., Mao, X.: Load balancing method for service scheduling of command information system. In: 2021 2nd Information Communication Technologies Conference (ICTC), pp. 297–301 (2021).

https://doi.org/10.1109/ICTC51749.2021.9441601

[42] Gulbaz, R., Siddiqui, A.B., Anjum, N., Alotaibi, A.A., Althobaiti, T., Ramzan, N.: Balancer genetic algorithm-a novel task scheduling optimization approach in cloud computing. Applied Sciences **11**(14) (2021). https://doi.org/10.3390/app11146244

[43] Al-Maytami, B.A., Fan, P., Hussain, A., Baker, T., Liatsis, P.: A task scheduling algorithm with improved makespan based on prediction of tasks computation time algorithm for cloud computing. IEEE Access **7**, 160916–160926 (2019). https://doi.org/10.1109/ACCESS.2019.2948704

[44] Kowsigan, M., Balasubramanie, P.: An efficient performance evaluation model for the resource clusters in cloud environment using continuous time Markov chain and Poisson process. Cluster Computing **22**(5), 12411–12419 (2019). https://doi.org/10.1007/s10586-017-1640-7

[45] Singh, A.K., Kumar, J.: Secure and energy aware load balancing framework for cloud data centre networks. Electronics Letters **55**(9), 540–541 (2019). https://doi.org/10.1049/el.2019.0022

[46] Kim, S.I., Kim, J.-K.: A method to construct task scheduling algorithms for heterogeneous multi-core systems. IEEE Access **7**, 142640–142651 (2019). https://doi.org/10.1109/ACCESS.2019.2944238

[47] Djigal, H., Feng, J., Lu, J.: Task scheduling for heterogeneous computing using a predict cost matrix. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops. ICPP 2019. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3339186.3339206

[48] Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: Distributed data-parallel programs from sequential building blocks. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. EuroSys '07, pp. 59–72. Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1272996.1273005

[49] Bondy, J.A., Murty, U.S.R.: Graph Theory. Graduate Texts in Mathematics, vol. 244, p. 651. Springer, New York (2008). https://doi.org/10.1007/978-1-84628-970-5. https://doi.org/10.1007/978-1-84628-970-5