Data-Flow Driven Optimal Tasks Distribution for Global Heterogeneous Systems

Jordi Garcia, Francesc Aguiló, Adrià Asensio,
Ester Simó, Marisa Zaragozá, Xavi Masip-Bruin

CRAAX Lab
UPC BarcelonaTECH
Vilanova i la Geltru, Spain

Abstract

As a result of advances in technology and highly demanding users expectations, more and more applications require intensive computing resources and, most importantly, high consumption of data distributed throughout the environment. For this reason, there has been an increasing number of research efforts to cooperatively use geographically distributed resources, working in parallel and sharing resources and data. In fact, an application can be structured into a set of tasks organized through interdependent relationships, some of which can be effectively executed in parallel, notably speeding up the execution time. In this work a model is proposed aimed at offloading tasks execution in heterogeneous environments, considering different nodes computing capacity connected through distinct network bandwidths, and located at different distances. In the envisioned model, the focus is on the overhead produced when accessing remote data sources as well as the data transfer cost generated between tasks at run-time. The novelty of this approach is that the mechanism proposed for tasks allocation is data-flow aware, considering the geographical location of both, computing nodes and data sources, ending up in an optimal solution to a highly complex problem. Two optimization strategies are proposed, the Optimal Matching Model and the Staged Optimization Model, as two different approaches to obtain a solution to the task scheduling problem. In the optimal model approach a global solution for all application's tasks is considered, finding an optimal solution. Differently, the staged model approach is designed to obtain a local optimal solution by stages. In both cases, a mixed integer linear programming model has been designed intended to minimizing the application execution time. In the studies carried out to evaluate this proposal, the staged model provides the optimal solution in 76% of the simulated scenarios, while it also dramatically reduces the solving time with respect to optimal. Both models have pros and cons and, in fact, can be used together to complement each other. The optimal model finds the global optimal solution at high running time cost, which makes this model unpractical on some scenarios. The staged model instead, is faster enough to be used on those scenarios; however, the given solution might not be optimal in some cases.

1. Introduction

Smart environments provide intelligent services aimed at facilitating users' everyday tasks while both economizing costs and enhancing sustainability. The main core technology in smart environments is based on the Internet of Things (IoT), which comprise a vast sensors network generating detailed information about any relevant aspect of the environment, all connected through low power wireless networks to a cloud data center where a rich set of smart services is processed in areas such as smart transportation, smart health, smart energy, or smart industry, just to name a few. These types of applications are becoming more complex and resource-hungry, demanding high performance computing and, sometimes, immediate answer. For this reason, these applications are usually executed at the cloud, which guarantees almost unlimited computation and storage capabilities, ubiquity, elasticity, and benefits from a cost efficiency model. However, as data becomes more prominent and bulky in modern smart applications, moving all the data to the cloud before computation poses severe inconveniences to this model, such as high communication latency, unnecessary network overload, high risks for failure as well as exposes data privacy [1].

Aimed at mitigating these limitations, fog computing [2] and edge computing [3] have been proposed as technological solutions to exploit the advantages of locality. By approaching computing devices to the data sources several advantages arise, such as enabling low latency, reducing data traffic at core networks, promoting green computing as well as guaranteeing higher privacy. Apparently, the computing capacity at the edge is much lower than that at the cloud; however, in [4] the authors estimated in 2018 the potential computing capacity in a major European city and showed to be 10 to 20 times the largest Amazon Web Service (AWS) data center. Similarly, in 2015, IBM Research forecasted that by 2017 the collective computing and storage capacity of smartphones would surpass all worldwide servers [5]. Although these figures illustrate the enormous potentiality at the edge, this technology has not been conceived to substitute the cloud, rather to complement each other and exploit the combined advantages of both technologies.

The scenario envisioned in this work is flooded with devices, from the edge up to the cloud, some of which are available to embrace offloaded computing and ready to execute applications on demand. In such a scenario, smart applications could be either executed close to the data sources, thus absorbing data transfer latency, or executed at cloud, providing enhanced computing power but with a communication penalty. Furthermore, when smart applications become more complex with geographically distributed multi-resource constraints, the solutions span becomes particularly challenging. For instance, some basic applications will be simply executed locally; alternatively, for many computing intensive applications the straightforward solution will be moving the computation to the cloud. But, in cases where the application data is highly distributed, or generated at real-time in remote locations, or obtained through streaming from distributed sources, and where several computing nodes are geographically distributed and available, solutions which efficiently exploit data locality can dramatically improve the execution performance.

The applications considered in this research work are large-scale scientific computing applications using distributed data sources, often comprising several interdependent workflows which can be decomposed into dependence constrained tasks [6]. Mapping such set of interdependent tasks into computing resources in heterogeneous systems is an interesting problem of current research. In the scope of cloud computing, existing container scheduling products cannot manage efficiently concurrent multi-resource requests when systems are heterogeneous [7]. Furthermore, this problem becomes far more challenging in an open context dealing with a large number of heterogeneous devices distributed through the edge, considering the physical devices' location and distance between them and the distributed geography of data sources and their connectivity to the system.

In this paper a data-flow driven optimal tasks to resources mapping technology for global heterogeneous management systems is presented. Specifically, given an application instance execution request which consists of a set of related tasks, given a set of physically distributed heterogeneous devices and their respective performance features, and given a set of geographically distributed data sources, this technology decides, statically, the best tasks to resources mapping to optimally execute that application in terms of execution time. In the proposed performance estimation model, for any tasks parallelization strategy, the different costs of executing the tasks on different nodes are estimated, including the task offloading penalty. In addition, the cost (latency and transfer times) of accessing the distributed data sets is also considered, both static data sources and dynamic data flow between running tasks, as well as any eventual data replication and network congestion. The novelty of this approach is that, to the best of the authors' knowledge, this is the first work to consider not only the heterogeneity and distribution of the computing nodes, but also the geographically scattering of data sources as well as the effects of the run-time data-flow for the selected parallelization strategy. In order to find a solution for this more realistic, but also more complex, scenario, two different optimization strategies have been considered based on mixed integer linear programming: the Optimal Matching Model (OMM) which finds a global solution for all tasks in the application, and the Staged Optimization Model (SOM) which finds local solutions by stages. The OMM model finds an optimal solution for the global problem, but when the number of tasks or the number of nodes is high, the time to find a solution becomes impractical. The SOM model is fast enough to be used in real scenarios; however, in some cases, the solution may not be optimal. Although in some cases the OMM model requires too much time for completion, it has been used in this paper to validate the quality of the solutions provided by the SOM model. By comparing the solutions provided by both models, the SOM model provides optimal solutions in most cases, and near-optimal solutions with a gap difference below 10% compared to the OMM. In both cases, the time saved by using the optimization strategy is from 36% to more than 80%.

The work presented in this paper is part of the Progressive Mapping System (PMS), a framework for global resources management designed to monitor millions of heterogeneous devices spread throughout some sort of broad smart environment to provide efficient applications execution for optimal performance [8]. Specifically, the technology presented in this paper focuses on the tasks to resources matching and has been designed to minimize the application execution time. Note that other metrics could be considered for minimization as well, such as energy consumption, quality of service, network load, or any other relevant metric. Finally, note that resources' mobility, volatility, or task migration, have not been considered in this research work. A static snapshot for the set of available resources is assumed, as well as an application instance with a static and known set of tasks. Considering dynamicity in the system opens a new dimension in the scheduling problem, which will be considered as future work and hence, out of the scope of this paper.

The remainder of this paper is organized as follows. In Section 2 the related work is discussed. Section III describes the problem to solve illustrated with an example scenario. In Section IV the problem requirements and specification are discussed, as well as the performance estimation cost model. Section V presents the two models for optimal tasks to resources matching, OMM and SOM. Section VI shows the numerical results that validate the mathematical models and, finally, Section VII presents the concluding remarks.

2. Related Work

Global resources management has long been an important research topic for the academic community and industry. Indeed, a large number of research contributions may be found in the literature aimed at cooperatively using a large set of geographically distributed resources set, organized and conceived as a single powerful computer, such as metacomputing [9], world wide virtual computer [10], global computing [11], volunteer computing [12] or, more recently, grid computing [13]. These technological proposals have been predecessors of the current cloud computing paradigm [14], which provides flexible dynamic infrastructures, quality of service guaranteed computing environments, and configurable software services, through the management and orchestration of huge amounts of computing facilities. With the recent advances in cloud computing and the increasing capabilities of mobile devices, compute-intensive processes can now run at the edge, on a set of mobile devices with highly limited computational capabilities, emerging the concept of mobile-cloud computing [15]. This is achieved by using the communication facilities of mobile devices to establish high-speed connections to larger computational resources located at cloud. In the same direction, cloudlets and mobile-edge cloud computing have been proposed as promising solutions intended to extend the utility of mobile-cloud computing to provide more powerful compute and storage resources accessible at the edge of the network [16]; and the EdgeCloud, which is a distributed management system for resource continuity in edge to cloud computing environments [17].

The authors of this paper also have previous experience in the design and implementation of global resources management systems with the goal of exploiting the available computing capabilities of edge systems. For instance, the Fog-to-Cloud (F2C) framework is a distributed system proposing a coordinated management of the whole set of resources within the cloud continuum spectrum, i.e., from the edge up to the cloud, building up a multilayered hierarchical coordinated structure [18]. Or, the Progressive Mapping System (PMS), a framework for massive resources management which performs task to resources mapping and allocation through progressive slicing techniques [8]. In the context of the PMS, a load distribution model for single-task fully parallel applications was also proposed [19]. And more recently, the authors proposed an efficient clustering strategy to optimally handle the whole set of systems at the edge through clustering policies, turning into much powerful edge instances [20].

In all these environments, a key critical control task to be optimized is an efficient distribution of the whole set of computational loads into the available devices. This problem becomes still more complex when devices are heterogeneous, as considered in this paper, what is absolutely normal in the envisioned systems, where cloud, edge and IoT devices are globally deployed. As systems scale up, applications in the smart era are ever more sophisticated and complex, leveraging all capabilities driven by technological advances, requiring intensive computational and communication resources, high energy consumption and, most importantly, consuming data spread throughout the whole smart environment. Applications can be usually organized into a set of precedence-constrained tasks, which can be modelled by directed acyclic graphs (DAG) [21], in which nodes represent the tasks to be executed and edges represent data communication between them. Placing independent tasks on different devices and running them in parallel can speed up computing and, therefore, improve the user experience. However, this can raise additional delay due to communication and synchronization overhead between physically distributed devices. Consequently, placing coordinated tasks on well connected devices may reduce this delay. Finally, a last consideration refers to the fact that tasks in smart scenarios use to require data generated (or perhaps, stored) in sensors (or their hosting device), which are geographically distributed. Similarly, placing tasks close to the data sources dramatically reduces communications costs and notably increases performance. For all these reasons, effective resource management and task scheduling mechanisms are required to improve the application performance in such smart environments. This is actually the main objective of the task scheduling problem, namely the design of strategies to allocate tasks in devices which minimize computation time as well as their offloading costs, and this is indeed the main research focus in this paper.

Multiple research works have dealt with the task scheduling problem in the scope of cloud-fog environments in recent years. Most studies have proposed heuristic solutions due to the problem complexity. Indeed, Wang and Li [22] established a smart production line simulation and proposed a task scheduling strategy based on a hybrid heuristic algorithm to minimize delay and energy consumption. Hoang and Dang [23] proposed a fog-based region architecture for provisioning nearby computation resources and designed an efficient heuristic algorithm to allocate tasks among regions and remote clouds. Xu et al. [24] proposed a scheduling algorithm based on particle swarm optimization, an intelligence evolutionary computing technology for seeking the trade-off between makespan and cost in a cloud-fog environment. Abdel-Basset et al. [25] proposed an energy-aware model to tackle the task scheduling problem in a fog computing environment. They proposed a marine predators algorithm as a heuristic to improve the quality of service required by users. In a different work, Abdel-Basset et al. [26] designed a metaheuristic algorithm for tackling task scheduling of Industrial IoT applications in fog computing. In that work, they used a Harris hawks optimization algorithm based on a local search strategy to improve the quality of service provided to the users. However, the proposed model is limited to schedule independent tasks. Hussein and Mousa [27] presented an efficient task offloading strategy to minimize task response times in a fog computing environment, driving two evolutionary meta-heuristic algorithms, leveraging ant colony optimization and particle swarm optimization.

Nevertheless, although the search for heuristic solutions is the most common way of tackling the task scheduling problem, different mathematical methods have also been proposed to find optimal solutions to this problem. Liu et al. [28] proposed an algorithm based on dynamic programming to optimally solve the dependent task placement and scheduling problem when the edge server configuration is fixed and known. Zeng et al. [29] proposed a mixed-integer nonlinear programming problem to minimize task completion time in fog computing systems. Barros et al. [30] proposed a solution to the context-aware mobile application task scheduling problem for fog computing. To define the priority of requests, multiple linear regression analysis was used and the optimal scheduling of tasks was solved using the multi-objective non-linear programming optimization technique. Additionally, Tran et al. [31] proposed an approach to task placement on fog computing for IoT application provisioning. The authors designed an optimization problem and implemented a program in cplex to solve the problem. However, in that work, the dependency between tasks was not considered because the tasks composing an application ran in parallel. From a mathematical perspective, some parallelism could be established with the task scheduling problem in multiprocessors systems. In this area, several studies that propose optimal scheduling can also be found. For instance, in [32], Venugopalan and Sinnen proposed a mixed-integer linear programming solution to the task scheduling problem, assuming that the communication links are identical. They used problem-specific knowledge to eliminate bilinear forms arising from communication delays, which reduced the complexity of the problem. In [33], Valouxis et al. proposed an integer programming model to minimize execution times for computer applications in a multiprocessor environment. They suggested distributing DAG nodes into levels and tackling the optimization by stages for DAGs of big size; however, they did not implement their proposal.

Compared to the related works, the novel contributions of this work are the following. A task placement mechanism for heterogeneous environments is proposed, which considers nodes with different computing capability, connected

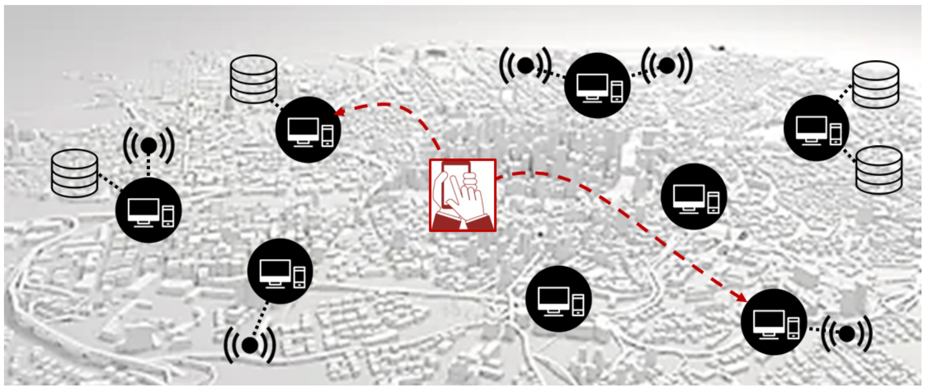

Figure 1: Illustrating example of the considered scenario.

through different network bandwidths, located at different distances and, the most interesting, considering the physical location of the all data sources as well as the data flow between tasks. Furthermore, a procedure has been designed to organize tasks hierarchically by levels based on the application's DAG, so that each level contains tasks that can be executed in parallel. Based on this hierarchical tasks distribution, two different optimization strategies have been designed to obtain a solution to the task scheduling problem and, in both cases, a mixed-integer programming model has been designed for minimizing the DAG optimization execution time, globally solving the problem.

3. Problem Description

In this research work, a complex technological scenario is envisioned with many heterogeneous computing devices, some of which storing data (historical repositories) and some of which attaching sensing devices (constantly generating streaming data), all connected through diverse communication technologies. In such scenario, one node in the system receives a request for application execution and the system must decide where it should be executed according to some performance metrics, for instance, execution time. In Fig. 1 an example of a smart city scenario is illustrated. In the figure, several computing nodes are shown, some of which host data repositories and/or sensing devices. If one application (smart service in this case) must be executed, it can be run either locally or offloaded to any of the available devices, possibly those which host the application's required data, or some other device close to them.

The application can be structured as a set of tasks organized through dependency relations. Each task could eventually require some data sets which are stored in or collected from one, or several, nodes in the system (static data), and could also generate, at run-time, additional data flows (dynamic data) which, in turn, could be used by the following tasks. Both, the computation cost and data sizes, are assumed to be known at launching time. Similarly, the topology and connectivity of the system available nodes is assumed to be known, including nodes' type and performance features, the connectivity technology, and the geographical location of nodes to be able to estimate the network delay. It is also known what data sources has initially each node in the system.

In this context, the problem statement is as follows. Given an application launching and given a system resources description and topology, find the best set of nodes to execute the application which minimizes the execution time, and determine where each application's task is executed. This problem has to be solved considering the computational cost, according to the tasks' load and nodes capacities, as well as the data flow communication overhead, according to the location of both the data sets (static data) and the nodes that execute the tasks and, consequently, produce and consume the run-time data (dynamic data).

Note that this problem poses several solutions. The trivial solution is to execute the application in the launching node (local execution). In this case, the performance depends on the node's performance and the volume and location of the required data sources. An alternative solution would be to execute the application at cloud. This solution potentially provides maximal performance (assuming you have almost unlimited computing capabilities at cloud), but data will likely be far away. The optimality of this solution depends, again, on the volume and location of the required data sources. Differently, another solution would be to execute the application in a node either owning the data or a large portion of them. In this case, the overhead of data movement will be drastically reduced and the performance will depend on the node's performance and availability. Finally, a span of alternative intermediate solutions should be appraised as well, considering any candidate node which trades-off computing power with locality features.

Lastly, it is worth mentioning that the envisioned scenario becomes much more complex when considering applications structured as a set of interdependent tasks, some eventually executed in parallel. In this case, all tasks can be executed either in a single node or distributed among several nodes to exploit the advantages of parallel execution. Note that the location of data sources has to be considered, as well as the dynamic data flow between tasks, which in turn depends on the geographical location of nodes selected for each task execution. Now, the span of candidate solutions grows exponentially.

The metric used for optimization in this current implementation is execution time, considering mainly both computation and communication times in a highly heterogeneous and distributed environment. Several performance features have been considered in the model, such as start-up time, network latency, bandwidth and capacity, nodes' distance (in number of hops), and so forth, which will be discussed in detail in the next section. However, alternative performance metrics could eventually be used, such as network traffic reduction (independently of the execution time), energy consumption, quality of service, resources rental time (in a pay-per-use environment), some sort of capital or operational expenses (CAPEX or OPEX) accounting system, or any other measurable or quantifiable metric.

4. Problem Requirements and Specification

In this section, the application and resources assumptions and requirements are described in detail, as well as the current cost model based on the application's execution time estimation.

4.1. Resources Requirements and Specification

Even though this work has been designed as part of a system for massive resources management, where the number of nodes considered can be potentially huge, in the PMS there is a filtering process based on the application features, such as the instance locality and run-time specification. This means that, in practice, the number of nodes considered in this optimal task to nodes matching will be of the order of several tens. Further details about the devices management and graph reduction modules, together with the discussion of some related open research challenges, can be found in [8]. It is also out of the scope of this work how the nodes discovery and availability is monitored. In fact, in this work a static set of nodes is assumed to be known and available to execute any task, and that this availability is constant through all the application execution. Addressing robustness, volatility and mobility in distributed systems is a challenging topic and, undoubtedly, should be considered in future stages of this research.

The system resources are described through the System Resources Graph (SRG), where nodes represent computing devices and edges represent nodes connectivity. Each device's computing power is tagged as a weight in the corresponding node and represents a percentage of relative performance with respect to one reference node. For instance, a node weighted with 100 yields a performance similar to the reference node and a node weighted with 80 yields an approximate performance of 80% of the reference node. Note that hardware details, such as processor or memory and caches features, have been omitted from the model, defining a weighting mechanism which is clearly a simplification of the reality; however, it allows having a simple yet effective model, trading-off accuracy with feasibility.

With respect to the connectivity, the SRG tries to reflect any communication relationship between nodes in the system, including network bandwidth and latency. As all nodes have access to Internet (otherwise they could not collaborate in a distributed application execution), all nodes are in fact connected to each other, posing high complexity in the system description (quadratic with respect to the number of nodes). In order to simplify this issue, a bandwidth feature is assigned to each node so that bandwidth between a pair of nodes is estimated as the minimal bandwidth of both nodes. For instance, if a node is connected to Internet through 4G with an average bandwidth of 10Mbps and another node is connected through a 3Mbps WiFi network, then a 3Mbps bandwidth between both nodes is estimated. Such simplification avoids having to measure (and specify) each bandwidth between any pair of nodes. Furthermore, assigning the bandwidth to the node allows estimating easily the channel saturation in case one node has to receive data from more than one node at a time, which is also considered in this model.

Latency is a performance metric proportional to the distance between nodes and highly depends on the nodes' physical location. However, it is not only dependent on the physical location but also on how they are connected and the physical routing between them. Accurately estimating latency is highly challenging and it is not the purpose of this model. For this reason, the latency between nodes is simplified by considering a delay proportional to the number of hops. Similarly, to avoid specifying each pair of individual nodes distances, nodes are grouped according to proximity (approaching a real scenario where nodes are physically grouped through their network provider access point). Then, the edges connecting groups of nodes are just weighted with some distance between groups measured in number of hops. The distance between nodes inside a group is considered to be one single hop.

Finally, the SRG also contains information about data sources (either files or sensors), which are assigned to nodes. Each node can own some data sources, and one data source can be replicated in one or more nodes (or, similarly, the same type of data can be collected from different sensors attached to different nodes).

For instance, Fig. 2 illustrates one possible SRG. The figure represents three groups of nodes with five, three, and four nodes, respectively. For simplicity, the nodes' tagging is only shown for nodes $N7$ and $N12$. The $N7$ node's computing power is weighted with 100, meaning a performance similar to the reference node, and its bandwidth has been labelled to 6Mbps. Also, $N7$ owns data sources $F1$, $F2$ and $F3$. Similarly, the $N12$ node's computing power is weighted with 70, meaning an approximate performance of 70% of the reference node (this could be a laptop), and bandwidth of 4Mbp. It owns data sources $F1$, $F4$ and $F5$. The SRG also contains information about the distance between groups of nodes, being 5 hops between group 1 and group 2, 3 hops between group 1 and group 3, and 2 hops between group 2 and group 3. Remember that the distance between nodes within the same group is assumed to be one hop.

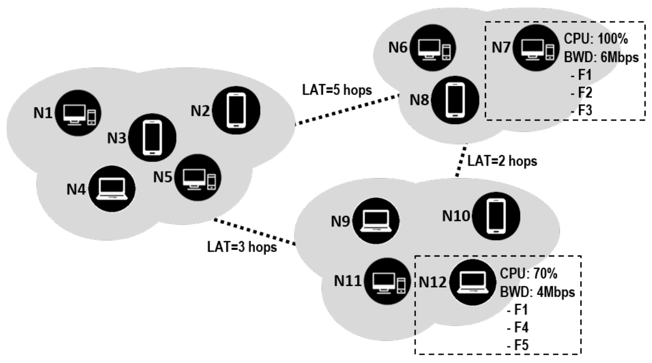

Figure 2: Illustrating example of a system resources graph.

All weights and related parameters in the SRG are currently being obtained empirically through profiling of all devices candidate to be part of the system. With these measures, a catalog of device types and weights is created, as well as the network interconnection technology. In future research, a classification of devices through supervised machine learning processes is planned, thus facilitating the process of resources categorization.

4.2. Application Requirements and Specification

The applications considered in the proposed optimization model consist of one or more tasks, requiring data hosted in several system nodes. This is a typical scenario in smart cities, for instance, where smart services use data generated from several sensors spread in different city locations. The number of tasks in the application is fixed and known, and eventually, some of these tasks can be executed in parallel (static task-level parallelism). In this case, a data dependency graph specifies dependency relationships between tasks. This type of applications is different from dynamic task-level parallelism, where a variable number of tasks are created at run-time according to the program environment.

Each task requires a specific (and generally known) data set (input data). Input data are, either stored (files) in one or more than one system nodes (replicas) or generated on-the-fly through some sensor devices (streaming data) likewise hosted in one or several system nodes. Tasks eventually produce new data sets (output data) which in turn can be consumed by other tasks (as input data). Note that in such highly distributed scenario, data-flow driven tasks distribution is a complex problem which should consider: i) the individual performance and connectivity of each node in the system; ii) the owners of all data sources required by the application, and; iii) the tasks' placement for those tasks that produce and consume the new data sets.

To be able to estimate the effects of distributing tasks over different nodes, tasks and data sets must be appropriately weighted. For this reason, one application execution instance is specified through a tasks' DAG, where nodes represent tasks and edges indicate dependencies between tasks. Nodes in the DAG are weighted with the execution time, measured in an average computer (recall that nodes in the system resources graph are weighted as a proportion of their performance concerning the average computer). Besides, nodes also have information about the input and output data sets. The input data is a list of data sets (stored files, sensing data, produced data), and the output data is the list of produced data in the corresponding node. Edges in the DAG, as explained earlier, represent the task dependencies of the application, where dependencies are key to define distinct execution orders which must be preserved, as well as to express some eventual parallel features which can eventually be exploited.

Finally, note that an application instance will be launched from one specific node in the system. This fact adds one constraint to the problem: executing the application, or some of its tasks, in a remote node should consider the cost of remotely moving the tasks or, at least, launching a remote method invocation. Even though this cost will presumably be quite low, it must be considered to favor local executions.

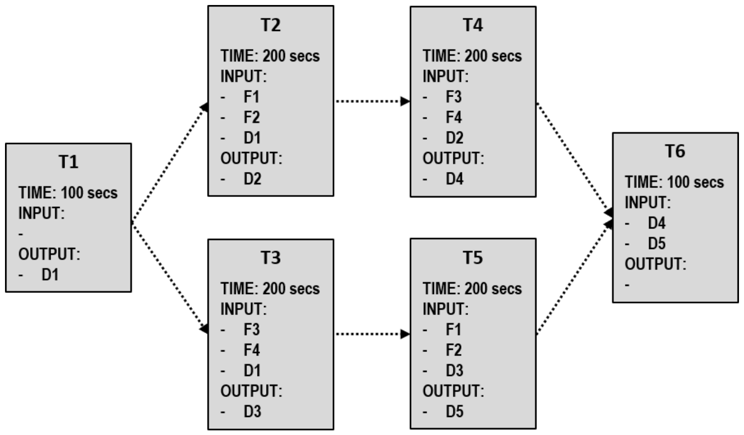

Figure 3: Illustrating example of a task graph with 6 tasks.

For instance, Fig. 3 illustrates one possible DAG instance. In this example, the application is decomposed into six tasks, which are organized through the dependencies graph. As part of the tasks' description, the task execution time allows estimating the weight of the task. Additionally, each task contains information about the data sets required (input data) and data generated dynamically (output data). For instance, $T2$ uses the data sources $F1$ and $F2$ and the dynamically generated data $D1$ (generated by $T1$), and generates $D2$ as output data. This information allows selecting which nodes own, or are close to, the required data sources. Furthermore, the system should consider which nodes execute each task, connecting (approaching) those nodes that dynamically produce data to nodes that consume them. For instance, the node that executes $T2$ should be close to the node that executes $T4$, or the node that executes $T3$ close to the node that executes $T5$. Finally, the DAG also reflects which tasks can be executed in parallel, for instance, tasks 2 and 3, or tasks 4 and 5.

Similar than in the previous subsection, this information is currently obtained empirically through profiling. However, some recent works from our group are paving the way to estimate the application's performance through supervised machine learning, in this case, in the scope of a fog-to-cloud framework [34].

4.3. Cost Model

Given an application's DAG, the set of resources from the SRG minimizing its execution time for that specific application must be selected. In order to measure the execution time for the different resources configurations, the cost model has to consider computation and communication times.

The computation time is estimated considering the tasks to resources allocation. Remember that tasks are weighted with a computation time measured in a reference computer, and computing devices are weighted with their relative performance with respect to the reference computer. Therefore, the computation time for an individual task in a given computing device is estimated by scaling the task's computation time with the computing device's relative performance. And the total computation time is estimated by traversing the DAG and considering the following cases: tasks with dependencies are executed sequentially, independent tasks executed in a single computing node are executed sequentially, and independent tasks executed in different computing nodes are executed in parallel, overlapping their estimated computation times. Finally, considering that the application is launched in a specific node, a small delay is added if the application is executed in a different node to reflect the latency for remote execution. Although this delay is irrelevant in most applications (30 ms, obtained as an average ping time), it allows the launching node to be favoured in case of equal conditions.

The communication time is estimated assuming both, the time to access the data sources (file systems or streaming data) and the data flow generated and consumed dynamically by the running tasks. The time to access a data source is estimated as follows. Given a node which owns a data source, and a node which needs the data source, the time to access the data source is estimated as the latency plus the communication time. The latency is computed as the distance between the two nodes (specified in the SRG as number of hops) multiplied by 30 ms (obtained as an average one-hop ping time), and the communication time is computed as the data volume (specified as part of the application instance description), divided by the bandwidth between the two nodes (assumed as the minimal bandwidth of both connecting nodes). Note that a data source could be in more than one node (replication) so, in this case, the selected source node will be the one which provides higher efficiency. The time to move data between running nodes is estimated similarly. Assuming the two nodes that execute related tasks which generate and consume a certain volume of data, the time to move the dynamic data flow is estimated as the latency (proportional to the nodes' distance), plus the communication time (proportional to the dynamic data volume and bandwidth between both nodes). Furthermore, accessing any local data set is assumed to be cost-less. Finally, in this cost model, an eventual network saturation has been considered when a node has to receive data from several other nodes. In this case, when the receiving node has a bandwidth higher than that of the sending nodes, then it can receive from several nodes in parallel as long as the aggregated bandwidth is lower or equal than that of the receiving node. Any remaining data will be received in successive data transfers.

Note that this cost model does not try to deliver an accurate performance prediction. This is just a reasonable approximation for an extremely complex problem, which considers sequential and parallel computation, data sources access communication time and latencies including the presence of replicas, dynamic data flow communication between running tasks, as well as an eventual network saturation. The goal of the performance optimization module is to provide an optimal tasks to nodes matching in a fast time. For this reason, the trade-off between some loss in accuracy (through some execution model simplification) and the feasibility of the solver has been considered.

5. Optimal Tasks to Resources Matching

As mentioned earlier, an application is decomposed into a set of tasks with dependency relationships, modelled through the tasks' DAG. A DAG is a digraph $G = (V, E)$ where vertices represent tasks ($V = \mathbb{T}$) and the set of directed edges E is defined by ordered pairs $(T_i, T_j) \in E$, with $T_i, T_j \in V$, such that T_j depends on T_i, i. e., the directed edges represent dependency relationships between tasks. Each vertex T_j of the DAG is assigned a label, indicating the workload (M_j) to be carried out, as well as information about the input data set (data files $\mathbb{F}$) and output (output data sets $\mathbb{D}$) of the task in question.

A static set of heterogeneous devices is assumed to be available to execute any task, represented through the SRG. The SRG is a resource graph $G' = (V', E')$, where vertices represent computing devices ($V' = \mathbb{N}$) and edges $\{N_i, N_j\}$ represent connectivity between devices. Each vertex N_i of the SRG has assigned a label, indicating the computing capacity (a_{i2}) of the device as a percentage of the relative performance with respect to a reference node, as well as the relation of data sources (data files $\mathbb{F}$) that are hosted on the device, and a label indicating the bandwidth assigned to the node N_i (B_{N_i}). An each edge $\{N_i, N_j\}$ of the SRG is labelled with the distance between the devices (in number of hops) and the bandwidth ($B_{N_iN_j}$) between nodes.

Static task scheduling is a complex problem to consider, which has been demonstrated to be NP-hard [35]. Similar than in other related works, as described in Section 2, mixed-integer linear programming technology is used to solve the optimization problem. Two different strategies have been designed to model this problem. The first strategy is OMM, which targets the task scheduling as a global model. In this case the solution obtained is optimal, but the complexity could become unfeasible for large problems. The second strategy is SOM, which uses a greedy algorithm based on a combinatorial process to find a local optimal solution by stages. In this case, solutions obtained are an upper bound for the OMM solution, but the solving time becomes fast enough to be used in real scenarios. Both strategies require the distribution of the tasks DAG by levels so that only tasks without dependencies are located at each level. This undoubtedly facilitates tasks to run in parallel, thus reducing the application execution time. In the next subsection, how the DAG is distributed among stages is described, which is necessary for both strategies, OMM and SOM. Afterwards, the OMM and SOM problem formulation are deeply described.

5.1. DAG Stages Distribution

In this section, given the DAG G of an application, the procedure to establish a distribution of tasks by levels is described. To this end, the set of tasks $\mathbb{T}$ is partitioned into task independent sets, that is, $\mathbb{T} = \bigcup_{s=0}^{S} \mathbb{T}_s$ such that $\mathbb{T}_s$ contains tasks that can be executed in parallel at stage s.

These sets of tasks $\mathbb{T}_0, \mathbb{T}_1, \ldots, \mathbb{T}_S \subset V$ are defined in such a way that:

(a) $\mathbb{T}_i \cap \mathbb{T}_j = \emptyset$ for $i \neq j$,

(b) no task in $\mathbb{T}_0$ is a depending task,

(c) each task in $\mathbb{T}_s$ depends on some task in $\mathbb{T}_{s-1}$ for $s \geq 1$,

(d) there is no task that depends on any task in $\mathbb{T}_S$.

Given a vertex T of the DAG, let us consider the following sets of vertices.

$$\Gamma^+(T) = \{T' \in V/(T, T') \in E\} \text{ and } \Gamma^-(T) = \{T' \in V/(T', T) \in E\}$$

The partition $\bigcup_{s=0}^{S} \mathbb{T}_s$ is obtained from the DAG G using the following algorithm:

1. The set $\mathbb{T}_0$ contains those tasks T such that $\Gamma^-(T) = \emptyset$.
2. Assign $s \leftarrow 0$.
3. Assign the working set $\mathbb{T}_w \leftarrow \mathbb{T}_s$.
4. Assign $s \leftarrow s + 1$.
5. Set $A = \bigcup_{T \in \mathbb{T}_w} \Gamma^+(T)$.
6. Set $B_r = \mathbb{T}_r \cap A$ and assign $\mathbb{T}_r \leftarrow (\mathbb{T}_r \setminus B_r)$ for each $r = 1, \ldots, s-1$.
7. Assign $\mathbb{T}_s \leftarrow A$.
8. If there is some vertex T in $\mathbb{T}_s$ having $\Gamma^+(T) \neq \emptyset$, then go to step 3).
9. Otherwise stop and output the sets $\mathbb{T}_0, \ldots, \mathbb{T}_S$.

Note that step 6) is not executed when $s = 1$. In addition, the set A in step 5) is defined from $\mathbb{T}_w$ following a modified BFS process. The generation of the DAG structure has been proven to be an NP-complete problem in the worst case (see [6]).

5.2. The Optimal Matching Model

Aimed at solving the Optimal Matching Model (OMM) problem, the following sets and parameters are defined:

$\mathbb{T}$	Set of tasks.
$\mathbb{N}$	Set of nodes.
$\mathbb{Q}$	Set of data and files; i.e., $\mathbb{D} \cup \mathbb{F}$.
$\mathbb{S}$	Set of stages. Each task T_i is executed in one and only one stage.
$\mathbb{P}$	Set of partitions of each stage, defining data movement order within a given stage.
$\gamma_{T_i}^{s}$	Binary with value 1 if task T_i is executed at stage s; 0 otherwise.
$\rho_{N_i}^{q}$	Binary with value 1 if data file q is initially available at node N_i; 0 otherwise.
$\delta_{T_i}^{q}$	Binary with value 1 if task T_i requires data q; 0 otherwise.
$\varphi_{T_i}^{q}$	Binary with value 1 if task T_i generates data q; 0 otherwise.
R_q	Integer, representing the amount of data of data type q, in bits.
$\xi_{T_i N_j}$	Binary with value 1 if task T_i must be executed at node N_j; 0 otherwise.
$l_{N_i N_j}$	Continuous representing the latency between nodes N_i and N_j.
$\sigma_{T_i N_j}$	Continuous representing the penalty applied when task T_i is executed at node N_j.

$B_{N_iN_j}$	Continuous representing the available bandwidth between nodes N_i and N_j; in bps.
B_{N_i}	Continuous representing the node's capacity at a given time, in bps.
a_{i1}	Parameter representing the computation startup time of node N_i.
a_{i2}	Parameter representing the computational capacity of node N_i.
M_i	Parameter representing the amount of single instructions of task T_i.

In addition, the following decision variables have been defined:

$x_{T_iN_j}$	Binary. Value 1 if task T_i is executed at node N_j.
$y^{qs}_{N_i}$	Binary. Value 1 if data q is available at node N_i at stage s.
$z^{q}_{T_iN_j}$	Binary. Value 1 if task T_i uses data q from node N_j.
$u^{qsp}_{N_iN_j}$	Binary. Value 1 if data q is moved from node N_j to node N_i at stage s and partition p.
$e^{sp}_{N_iN_j}$	Continuous, accounting data move time between N_j and N_i at stage s, partition p.
$m^{sp}_{N_iN_j}$	Binary. Value 1 if data is moved from N_j to N_i at stage s, partition p.
$v^{sp}_{N_i}$	Continuous, accounting data access time for N_i at stage s, partition p.
w^s	Continuous, accounting maximum execution plus data transmission time at stage s.

Then, the mixed-Integer Linear Programming (MILP) model for the OMM problem is defined as:

$$Minimize\ OMM\ Cost \tag{1}$$

where the *OMM Cost*, or the objective function, is defined by:

$$OMM\ Cost = \sum_{s\in\mathbb{S}} w^s + \sum_{T_i\in\mathbb{T}} \sum_{N_j\in\mathbb{N}} \sigma_{T_iN_j}\cdot x_{T_iN_j} \tag{2}$$

and subject to:

$$\sum_{N_j\in\mathbb{N}} x_{T_iN_j} = 1\ \forall\ T_i\in\mathbb{T} \tag{3}$$

$$x_{T_iN_j} \geq \xi_{T_iNj}\ \forall\ T_i\in\mathbb{T}, N_j\in\mathbb{N} \tag{4}$$

$$y^{qs}_{N_i} = \rho^{q}_{N_i}\ \forall\ q\in\mathbb{Q}, N_i\in\mathbb{N}, s=0 \tag{5}$$

$$y^{q(s+1)}_{N_i} \leq y^{qs}_{N_i} + \sum_{T_j\in\mathbb{T}} (\delta^{q}_{T_j} + \varphi^{q}_{T_j})\cdot\gamma^{s}_{T_j}\cdot x_{T_jN_i}\ \forall\ q\in\mathbb{Q}, N_i\in\mathbb{N}, s\in\mathbb{S}\backslash s=|\mathbb{S}|-1 \tag{6}$$

$$y_{N_i}^{q(s+1)} \geq \frac{y_{N_i}^{qs} + \sum_{T_j \in \mathbb{T}} (\delta_{T_j}^q + \varphi_{T_j}^q) \cdot \gamma_{T_j}^s \cdot x_{T_j N_i}}{bigM} \;\forall\; q \in \mathbb{Q}, N_i \in \mathbb{N}, s \in \mathbb{S} \setminus s = |\mathbb{S}| - 1 \tag{7}$$

$$\sum_{N_j \in \mathbb{N}} z_{T_i N j}^q = \delta_{T_i}^q \;\forall\; T_i \in \mathbb{T}, q \in \mathbb{Q} \tag{8}$$

$$z_{T_i N_j}^q \leq \sum_{s \in \mathbb{S}} \gamma_{T_i}^s \cdot y_{N_j}^{qs} \;\forall T_i \in \mathbb{T}, q \in \mathbb{Q}, N_j \in \mathbb{N} \tag{9}$$

$$\begin{aligned} z_{T_j N_i}^q \geq z_{T_k N_i}^q - bigM \cdot (2 - \delta_{T_j}^q \cdot \gamma_{T_j}^s \cdot x_{T_j N_i} - \delta_{T_k}^q \cdot \gamma_{T_k}^s \cdot x_{T_k N_i}) \\ \forall\; s \in \mathbb{S}, N_i \in \mathbb{N}, T_j, T_k \in \mathbb{T}, q \in \mathbb{Q} \end{aligned} \tag{10}$$

$$\begin{aligned} z_{T_j N_i}^q \leq z_{T_k N_i}^q + bigM \cdot (2 - \delta_{T_j}^q \cdot \gamma_{T_j}^s \cdot x_{T_j N_i} - \delta_{T_k}^q \cdot \gamma_{T_k}^s \cdot x_{T_k N_i}) \\ \forall s \in \mathbb{S}, N_i \in \mathbb{N}, T_j, T_k \in \mathbb{T}, q \in \mathbb{Q} \end{aligned} \tag{11}$$

$$\begin{aligned} \sum_{p \in \mathbb{P}} u_{N_j N_k}^{qsp} \geq z_{T_i N_k}^q - bigM \cdot (1 - \delta_{T_i}^q \cdot \gamma_{T_i}^s \cdot x_{T_i N_j}) \\ \forall q \in \mathbb{Q}, s \in \mathbb{S}, T_i \in \mathbb{T}, N_j, N_k \in \mathbb{N} : N_j \neq N_k \end{aligned} \tag{12}$$

$$u_{N_i N_j}^{qsp} \leq m_{N_i N j}^{sp} \;\forall q \in \mathbb{Q}, s \in \mathbb{S}, p \in \mathbb{P}, N_i, N_j \in \mathbb{N} \tag{13}$$

$$m_{N_i N_j}^{sp} \leq \sum_{q \in \mathbb{Q}} u_{N_i N_j}^{qsp} \;\forall s \in \mathbb{S}, p \in \mathbb{P}, N_i, N_j \in \mathbb{N} \tag{14}$$

$$\sum_{N_j \in \mathbb{N}} B_{N_i N_j} \cdot m_{N_i N j}^{sp} \leq B_{N_i} \;\forall s \in \mathbb{S}, p \in \mathbb{P}, N_i \in \mathbb{N} \tag{15}$$

$$\begin{aligned} e_{N_i N_j}^{sp} \geq l_{N_i N_j} + \sum_{q \in \mathbb{Q}} \frac{R_q}{B_{N_i N_j}} \cdot u_{N_i N_j}^{qsp} - bigM \cdot (1 - m_{N_i N_j}^{sp}) \\ \forall s \in \mathbb{S}, p \in \mathbb{P}, N_i, N_j \in \mathbb{N} : N_i \neq N_j \end{aligned} \tag{16}$$

$$v_{N_i}^{sp} \geq e_{N_i N_j}^{sp} \;\forall s \in \mathbb{S}, p \in \mathbb{P}, N_i, N_j \in \mathbb{N} \tag{17}$$

$$w^s \geq \sum_{p \in \mathbb{P}} v_{N_j}^{sp} + \sum_{T_i \in \mathbb{T}} \gamma_{T_i}^s \cdot (a_{j1} + \frac{M_i}{a_{j2}}) \cdot x_{T_i N_j} \;\forall s \in \mathbb{S}, N_j \in \mathbb{N} \tag{18}$$

The objective function (1,2) minimizes the total time to execute the application. Equation (3) guarantees that all tasks are executed in one and only node; while equation (4) assigns a task to be executed in a particular node if it is required. Equations (5)-(7) deal with data availability at the nodes for each stage. Specifically, equation (5) ensures that data initially available at node N_i is available at the first stage, s=0. Equation (6) ensures that if data q is not available, moved or generated in a given node N_i for a given stage s, that data is not available in that node at the next stage. Equation (7) guarantees that if data q is available in a node N_i at stage s or if data is required or generated in that node, it will be available also in the next stage (data remains in the node). Equations (8) and (9) guarantee that each task accesses the data needed, and that only nodes where the required data is available can be selected as source of data, respectively. Equations (10) and (11) ensure data to be moved to a node is only moved from one and only one node (i.e., there is only one source of data for all tasks executed in the same node requiring that data). Equation (12) accounts if data q is moved between each pair of nodes for each stage. Equations (13) and (14) ensure that if no data is moved between two nodes at a given partition for a stage, no data is moved between that nodes at that stage, and vice versa, respectively. Equation (15) guarantees that the maximum bandwidth in the target nodes is not exceeded when moving data to them. Equation (16) accounts the maximum time to move data between pairs of nodes in each stage and partition of stage. Equation (17) accounts the maximum time to access data from a given node, stage and partition of stage. Finally, equation (18) accounts the total time required (data transmission time plus execution time) for each stage.

In terms of complexity, the OMM optimization problem is NP-hard in the worst case, since it is based on the static task scheduling problem, which was proved to be NP-hard [35]. Regarding its size, the number of variables is in the order of $\mathrm{O}(|\mathbb{Q}| \cdot |\mathbb{S}| \cdot |\mathbb{P}| \cdot |\mathbb{N}|^2)$ and the number of constraints is in the order of $\mathrm{O}(|\mathbb{Q}| \cdot |\mathbb{S}| \cdot |\mathbb{N}| \cdot \Big[2 \cdot |\mathbb{T}|^2 + |\mathbb{N}| \cdot |\mathbb{T}| + |\mathbb{N}| \cdot |\mathbb{P}|\Big])$. Considering a scenario based on a 30-nodes network, 6 tasks to be executed in 4 stages, 4 different data types and partitions, the number of variables and the number of constraints are in the order of 10^4 and 10^5, respectively; which makes the above mathematical model impracticable to be solved in short times. Indeed, increasing the number of tasks and nodes, rapidly makes the problem to be intractable, even using state-of-the-art computer hardware and the latest commercially available solvers.

Recognized the size and complexity of the OMM problem, in the next subsection, an algorithm to solve this problem in realistic scenarios is proposed.

5.3. The Stage Optimization Model

Aimed at solving the Stage Optimization Model (SOM) problem, first, a combinatorial procedure is defined to compute the minimum transmission time $\tau(i, j)$ of T_j's input data ($\mathbb{F}_j$ and $\mathbb{D}_j$) to node N_i. To this end, assume some independent tasks, $\mathbb{T}_s = \{T_{j_1}, \ldots, T_{j_{m_s}}\}$, have to be executed at some fixed stage s. The process from the point of view of task $T_j \in \mathbb{T}_s$ is described as follows. Consider:

- T_j has input data $\mathbb{F}_j = \{F_{u_1}, \ldots, F_{u_{\alpha_j}}\}$ and $\mathbb{D}_j = \{D_{v_1}, \ldots, D_{v_{\beta_j}}\}$ which are ordered sets. That is, file data of type u_l is at position l of the set $\mathbb{F}_j$. Note that $|\mathbb{F}_j| = \alpha_j$ and $|\mathbb{D}_j| = \beta_j$ for $\alpha_j, \beta_j \in \{0, 1, \ldots\}$.

- $F_{u_l} \in \mathbb{F}_j$ is located at nodes $\mathbb{N}^F_{l,s} = \{N_{f_1}, \ldots, N_{f_{G_l}}\}$. The subindex l of $\mathbb{N}^F_{l,s}$ stands for the position l in the set $\mathbb{F}_j$.

- $D_{v_k} \in \mathbb{D}_j$ is located at nodes $\mathbb{N}^D_{k,s} = \{N_{d_1}, \ldots, N_{d_{L_k}}\}$. The subindex k of $\mathbb{N}^D_{k,s}$ refers to position k in the set $\mathbb{D}_j$.

For each $F_{u_l} \in \mathbb{F}_j$ consider the set $V_l^F = \{(f_1, u_l), \ldots, (f_{G_l}, u_l)\}$, $l \in \{1, \ldots, \alpha_j\}$. For each $D_{v_k} \in \mathbb{D}_j$ consider the set $V_k^D = \{(d_1, v_k), \ldots, (d_{L_k}, v_k)\}$ for each $k \in \{1, \ldots, \beta_j\}$. The sets V_l^F and V_k^D contain the indices of nodes owning F_{u_l} and D_{v_k}, respectively. Let V_j be the Cartesian product defined by

$$V_j = V_1^F \times \cdots \times V_{\alpha_j}^F \times V_1^D \times \cdots \times V_{\beta_j}^D. \tag{19}$$

An element e of V_j is an $(\alpha_j + \beta_j)-$tuple, where each component of this tuple will be given by a pair ('node index', 'data type index'), that is

$$e = ((a_1, u_1), \ldots, (a_{\alpha_j}, u_{\alpha_j}), (b_1, v_1), \ldots, (b_{\beta_j}, v_{\beta_j})) \tag{20}$$

with $(a_l, u_l) \in V_l^F$ for $l \in \{1, \ldots, \alpha_j\}$ and $(b_k, v_k) \in V_k^D$ for $k \in \{1, \ldots, \beta_j\}$. The set V_j contains the indices of all nodes containing input data at stage s. All possible ways of obtaining input data when e ranges in the set V_j are obtained. Let $S(e)$ be the set whose elements are the components of e, that is $S(e) = \{(a_1, u_1), \ldots, (a_{\alpha_j}, u_{\alpha_j}), (b_1, v_1), \ldots, (b_{\beta_j}, v_{\beta_j})\}$. Let $\mathcal{P}(S(e))$ be the set of partitions of $S(e)$. Each partition $P \in \mathcal{P}(S(e))$ defines a way of transmitting the T_j's input data to node N_i. And all different ways of doing these transmissions are defined by some partition. Clearly, those partitions that allow parallel transmissions have to be prioritized. To this end, partitions $P \in \mathcal{P}(S(e))$ are considered with sets $A \in P$ of either $|A| = 1$ or, when $|A| > 1$, sets that fulfill the condition $\sum_{(p,q)\in A} B_{N_p} \leq B_{N_i}$. Let us denote the set containing this type of partitions by $\mathbb{P}_i(e)$, where index i corresponds to the node N_i pointed by the first argument of $\tau(i, j)$. The quantity B_N stands for the capacity of a generic node N.

The condition $\sum_{(p,q)\in A} B_{N_p} \leq B_{N_i}$ over the sets A with $|A| > 1$ in partitions of $\mathbb{P}_i(e)$ is necessary for parallel transmission when possible. At the same time, many partitions not showing optimal transmission time do not belong to the set $\mathbb{P}_i(e)$. Thus, the algorithm filters many transmission ways with related non-optimal transmission time at a given stage s.

Let $\theta(i, p, q)$ be the transmission time of F_q (or D_q) from node N_p to node N_i when $q = u_l$ (or $q = v_k$) for each $(p, q) \in A$, $A \in P$ and $P \in \mathbb{P}_i(e)$. This quantity is defined by

$$\theta(i, p, q) = \begin{cases} 0 & p = i, \\ l_{N_i N_p} + c_{ip} \cdot R_q & p \neq i, \end{cases} \tag{21}$$

where R_q is the amount of data of F_q (or D_q) and $c_{ip} = \frac{1}{B_{N_i N_p}}$.

Then, for $(p, q) \in A$, the transmission time of all input data contained in node N_p is given by

$$\phi(i, A) = \max\{\theta(i, p, q) : \ (p, q) \in A\}. \tag{22}$$

So, the minimum transmission time $t(i, j, e)$ for the data sets $\mathbb{F}_j$ and $\mathbb{D}_j$ to be transmitted from nodes in $S(e)$ to node N_i is

$$t(i, j, e) = \min_{P \in \mathbb{P}_i(e)} \sum_{A \in P} \phi(i, A).$$

Therefore, the minimum transmission time $\tau(i,j)$ is given by

$$\tau(i,j) = \min_{e\in V_j} t(i,j,e). \tag{23}$$

Note that there is some element $e^* \in V_j$ such that $t(i,j,e^*) = \tau(i,j)$ and those nodes which give this minimum value can be traced from the element e^*. Note also that, when applying this combinatorial algorithm, e^* is the first element found in V_j giving minimum transmission time. In fact, more than one element in the set V_j may give this minimum value.

Once all $\tau(i,j)$ have been calculated for a given stage s, the related linear problem intended to find which nodes execute tasks in stage s can be solved. Assume the scenario has n total nodes and the stage s has m_s tasks $\mathbb{T}_s = \{T_{j_1},\ldots,T_{j_{m_s}}\}$. Define the set of indices $J_s = \{j_1,\ldots,j_{m_s}\}$ from the set of tasks $\mathbb{T}_s$. Next, the variables and coefficients of the following MILP program are detailed:

$x_{T_iN_j}$	Binary variable. Value 1 if task T_i is executed at node N_j.
t_i	Continuous variable. Communication and computing time for node N_i.
t^*	Continuous variable. The maximum $\max_{1\le i\le n} t_i$. This variable is also the objective function to be minimized.
$\tau(i,j)$	Coefficient. Minimum transmission time of T_i's input data to node N_i.
a_{i1}	Coefficient. Computation startup of node N_i.
a_{i2}	Coefficient. Computational capacity of node N_i.
M_j	Coefficient. Amount of single instructions of task T_j.
m_s	Coefficient. Number of task at stage s.

The following linear problem gives the minimum time to run tasks in stage $\mathbb{T}_s$ and also it gives which node executes each task.

$$\text{Minimize } t^*$$

$$\text{under restrictions:}$$

$$\sum_{j\in J_s} x_{T_jN_i}\cdot\left(\tau(i,j)+a_{i1}+\frac{M_j}{a_{i2}}\right) = t_i, \quad (i=1,\ldots,n) \tag{24}$$

$$t_i \le t^*, \quad (i = 1\ldots,n) \tag{25}$$

$$\sum_{i=1}^{n} x_{T_jN_i} = 1, \qquad (j\in J_s) \tag{26}$$

$$\sum_{j\in J_s} x_{T_jN_i} \le m_s, \ (i=1,\ldots,n) \tag{27}$$

$$\sum_{i=1}^{n}\sum_{j\in J_s} x_{T_jN_i} = m_s. \tag{28}$$

Equation (25) sets minimum time t^* as the maximum value of t_i for $1 \leq i \leq n$. Equation (26) enforces task T_j runs on some node for each $j \in J_s$. Equation (27) imposes a node to execute m_s tasks at most. Equation (28) assures all tasks in $\mathbb{T}_s$ will be executed.

Once the selection of nodes by the linear problem is done, stage s ends. Then, data transmitted and generated by tasks is updated in the scenario. Next, a new stage begins. A solution to the SOM problem is obtained by adding all local–stages' optimal times. This solution will be an upper bound for the optimal global time.

Regarding the time complexity, the model needs to apply two algorithms at each stage. A combinatorial algorithm that filters many non-optimal transmission ways of input data between nodes and then, a MILP problem. Both algorithms have an NP-hard time complexity in the worst case [36]. The MILP problem needs the coefficients $\tau(i, j)$ which have been pre-computed by the above mentioned combinatorial algorithm. Furthermore, this problem has $n \cdot m_s$ binary variables, $n + 1$ continuous variables and $3n + m_s + 1$ restrictions. The filtering process of the combinatorial algorithm makes the SOM model light enough in practice.

6. Illustrative and Numerical Results

In this section, the scenarios considered to generate problem instances are described, including reference scenarios defining benchmark values. Next, the proposed OMM and SOM models are validated, and the performance results are compared in terms of cost and time to solve the problem, for both the OMM and SOM models as well as the considered benchmark reference scenarios.

6.1. Scenarios Description

In order to test the efficiency and effectiveness of the proposed models, three sample applications have artificially been created with different degrees of parallelism and dependencies. They consist of 1, 2, and 6 tasks, and are referred to as *App-1*, *App-2*, and *App-6*, respectively. *App-1* is a basic application with a single task, using four static files. *App-2* is an application with two dependent tasks using two static files each, and moving some dynamic data from the first to the second task. And *App-6* is the application with 6 tasks illustrated previously in Figure 3. Note that tasks $T2$ and $T3$ depend on $T1$, and that the couple $T2$ and $T4$ (executed in sequence) can be executed in parallel to the couple $T3$ and $T5$ (also in sequence). In addition, tasks 2 to 5 are using four static files, and all dependent tasks must transfer some dynamic data among them. Moreover, the number of partitions is set to 4, since it allows to transfer data and files between nodes in all the cases evaluated (thus avoiding infeasible problem instances), and defining more partitions results in increasing complexity, as shown in the complexity analysis for the ILP model. Table 1 summarizes the main characteristics of all applications. The size of the static files and dynamic data is 10MB each. The penalty applied when the application is executed in a different node than the launching node is 30 ms per hop, from the launching node to the node selected for the initial task execution.

Table 1: Description of the applications.

Parameter	**App-1**	**App-2**	**App-6**
Launching node	N1	N1	N1
Num. Tasks	1	2	6
Num. Stages	1	2	4
Num. Files	4	4	4
Max. Files per Task	4	2	2
Num. Data	0	1	5
Min. Exec. Time	100	300	600

In addition, based on the selected applications, a reference scenario is defined in which all tasks are executed locally in the node that launches the application execution. That is, no workload distribution is performed; therefore, these cases represent the base benchmark to compare with. These cases are referred as *App-1 BM*, *App-2 BM* and *App-6 BM*. The (ideal) case is also defined in which a given application is executed in the node it is launched at, assuming both the node performs as the best/reference node and all files required by the application are available in that node. It is worth noting that, in this case, the time to execute the application is the total time to finish each stage (assuming the best computing resources in the node). These cases are referred as *App-1 Min*, *App-2 Min* and *App-6 Min*.

Regarding the resources topologies, a set of different scenarios have been created varying the number of nodes from 5 to 50, and generating random SRGs. The process to generate each network topology is described as follows. Given a certain number of nodes NN, a random number of groups of nodes NG is generated (remember that nodes are grouped according to proximity, to be considered when computing distance, i.e., number of hops) by assuming an average of 8 nodes per group $NG = NN/8$ and then adjusting this value randomly as $NG = NG + rand(-NG/2, NG/2)$ (and making sure the number of groups is, at least, 1). Then, a random number of nodes is assigned to each group of nodes, making sure that the total number of nodes is finally NN. For each node, the computing power has been assigned randomly between 25%, 50%, 75% and 100%, as well as the bandwidth, which has been assigned from 1 to 4 Mbps. Furthermore, each node has a probability of 33% to own a file, 33% $\cdot$ 33% to own two files, and so forth (up to four files). Finally, distances between each pair of groups of nodes have been assigned randomly between 1 to 5 hops.

Both models, SOM and OMM, have been implemented in Python. To solve the problem instances for SOM, SageMath [37] was utilized; whereas the problem instances for the OMM problem have been solved using CPLEX [38]. A 64-bits computer built upon 12 Intel core ES-2620 v2 at 2.1GHz and 128 GB RAM has been used in all cases.

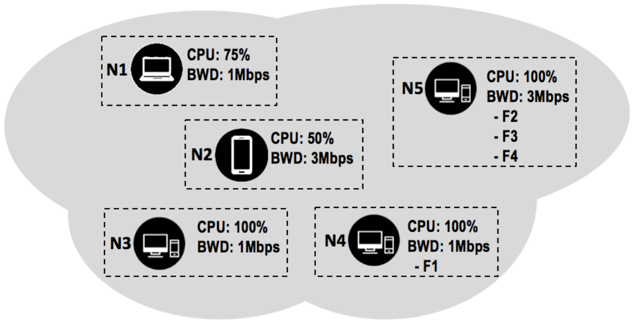

Figure 4: 5-node network topology.

6.2. Optimization Models Validation

Before comparing performance results of the proposed models, the OMM and SOM have been validated. To that end, first, problem instances based on the *App-1* described before (i.e., an application with a single task) are addressed. It is worth noting that, in these cases, the proposed models should find the optimal solutions. Next, an arbitrary problem instance based on *App-2* has been analyzed to illustrate the impact (on the cost) when SOM is considered against the optimal solution, obtained by OMM.

Consider *App-1* (see Table 1 for the summary of its main features), and the 5-node topology depicted in Fig. 4 . In this example, the solution that minimizes the execution time is to execute *App-1* at node *N5* and move file *F1* from node *N4* to it. After solving the corresponding problem instances for the proposed models, both models provide the optimal solution, as expected. In this case, the cost is 183.946 s; which is computed as the sum of the time to move file *F1* to *N5* (83.916 s), the processing time for the task at node *N5* (100 s) and the penalty (30 ms) due to executing the task in a different node that the launching node, *N1*. The time of executing *App-1* at the launching node is 468.937 s. That is, the application is executed in about 60% less time (284.991 s) when the workload is distributed according to the proposed models.

As previously described, for single-task applications both models provide optimal solutions. Fig. 5 shows the results obtained for solving problem instances with 5-node network topologies and *App-1* with the proposed models. In addition, the benchmark and ideal values are shown. Relative time savings of 86%, 55% and 60% are observed against the values obtained for the reference scenarios for problem instances numbered as 1, 4 and 5, respectively. It is worth recalling that, the network topology depicted in Fig. 4 has been considered in problem instance 5. For problem instances 2 and 3, the optimal solution is to execute the application at the same node it is launched. Moreover, although results obtained for instance 1 are very close to the ideal ones, it is worth noting that, for that instance node *N5* is the one having all required files and the best computing resources. The cost is 100.03 s, which includes the penalty added by executing the task in a different node than the launching node (*N1*).

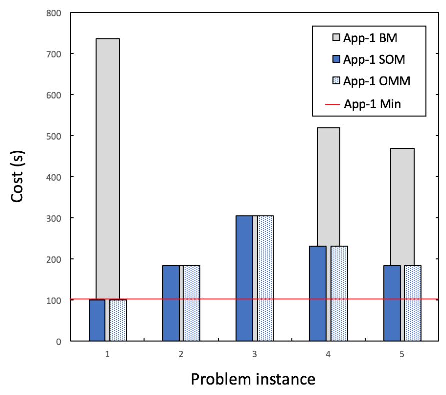

Figure 5: Cost against problem instance for *App-1* and 5-node network topologies.

For completeness, let us now focus on a specific problem instance in which the solutions obtained by the models are different; i.e., non-optimal solutions are obtained by SOM. An instance of the *App-2* (see Table 1 for details) and the 5-node network topology illustrated in Fig. 4 have been considered. In this case, identifying the optimal solution is not as obvious as in the previous example, shown for *App-1*. After solving the problem utilizing SOM, the solution obtained is to execute the first task at node *N4* and the second task at node *N5*; which implies that: *i*) at the first stage file *F2* is moved from node *N5* to node *N4*, and *ii*) at the second stage, data generated by the first task needs to be moved from node *N4* to node *N5*. The cost in this case is 467.862 s. However, this is not the optimal solution; which, in fact, is obtained when solving the problem utilizing OMM and consists in executing both tasks at node *N5*, thus moving only file *F1* from node *N4* to node *N5*. From the solution obtained, the optimal value is 383.946 s. The reasoning behind this is that the SOM aims at finding the best solutions for each stage in order, with no global knowledge about other stages. Differently, the OMM solves the problem globally; that is, considering the interrelation between tasks that are executed at different stages.

Interestingly, the value obtained when no workload distribution is considered (i.e., both tasks are executed at node *N1*) is 735.634 s. Therefore, considering the workload distribution to solve that problem instance results in time saving higher than 47% and 36% when OMM and SOM are utilized, respectively.

6.3. Performance results

After validating the proposed models, their performance is compared in terms of the solutions obtained and time to solve the problem; which, in the case of OMM, includes the time to generate problem equations in addition to the time to solve them and obtain the solution, after invoking the solver. To that end, several problem instances for *App-1*, *App-2* and *App-6* are generated.

Fig. 6 shows average cost values obtained for *App-1* (Fig. 6a), *App-2* (Fig. 6b) and *App-6* (Fig. 6c) against the number of nodes. Each point in the figure

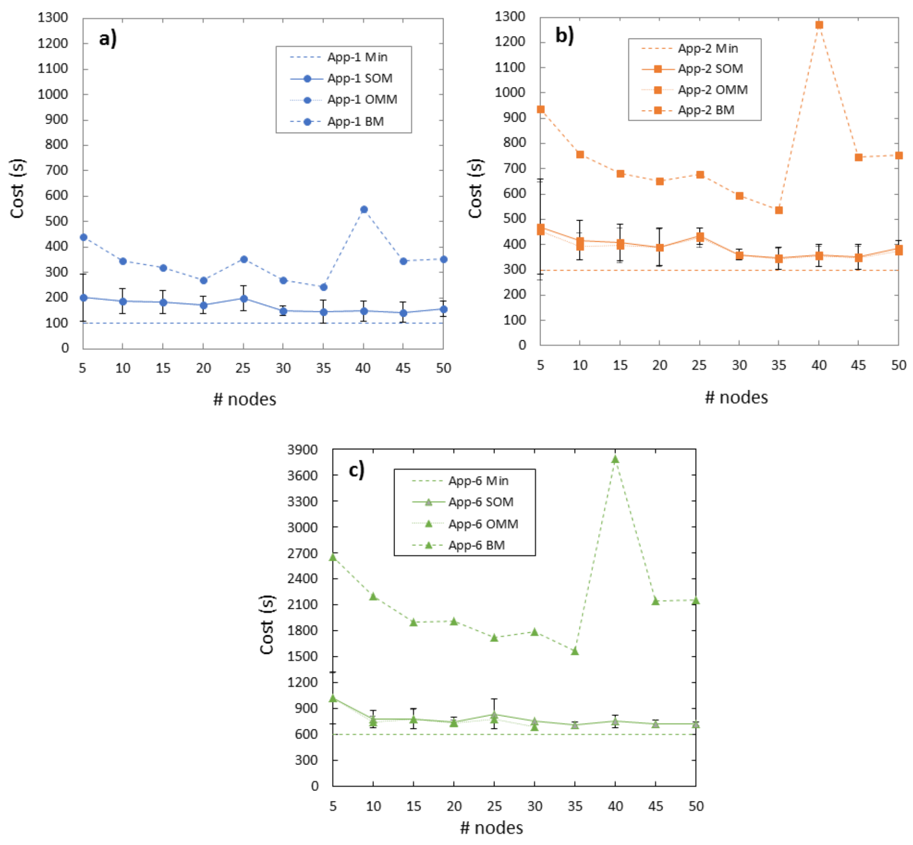

Figure 6: Cost against number of nodes for *App-1* (a), *App-2* (b) and *App-6* (c).

is the average value for 5 executions, corresponding to a different network topology and a fixed number of nodes ranging from 5 to 50, as previously described. For completeness, 95% confidence interval is also depicted for OMM and SOM lines. As expected, for *App-1*, results obtained for both proposed models are the same, i.e., optimal; therefore, in the figure, both OMM and SOM lines are overlapped and seen as a single line. Interestingly, as it can be observed in Fig. 6a, the proposed models significantly reduce the execution time of the application, compared to the benchmark scenarios, where no workload distribution is performed. Furthermore, the results depicted in Fig. 6b and Fig. 6c, corresponding to applications having more than one task (i.e., *App-2* and *App-6*, respectively), show that, on average, OMM performs slightly better than SOM for some *App-2* problem instances (e.g., see results for 5 and 10 nodes); whereas this difference increases for some *App-6* problem instances (e.g., see results for 25 and 30 nodes). However, as it can be seen in the figures, both OMM and SOM curves are quite close; thus showing that the solutions obtained by SOM are very close to the optimal ones. In addition, it is worth noting that, for

App-6 and 25 nodes, one instance could not be solved optimally in more than 2.5 hours; similarly, for *App-6* and 30 nodes, three instances could not be solved optimally (after more than 2.5 hours). Therefore, these instances have not been depicted in the results shown in the figures. In summary, the SOM is able to effectively find a solution for any problem instance and network topology, whereas the OMM eventually fails in providing an optimal solution in some extreme cases. In all cases, solutions obtained through SOM are almost identical to those provided by OMM, and clearly improve performance compared to the reference scenarios.

Tables 2 and 3 provide deeper insight about the gap of SOM against the optimal value, as well as the time saving (relative and absolute) of SOM against the benchmark values for *App-2* and *App-6*, respectively. As described above, SOM performs very similar to OMM. On average, the gap between the cost of the solutions obtained by SOM and the optimal ones is below 6% and even close to 0% for *App-2* and the number of nodes considered; whereas in the worst case (i.e., for *App-6* and 30 nodes), the gap is close to 10% and it is below 8% for *App-6* and any number of nodes up to 25. Although the gap for *App-6* when considering more than 30 nodes cannot be obtained in relatively short times, it is clear that the gap value will be between 0% and the gap to *App-6 Min* value (i.e., 600 s). In addition, SOM clearly outperforms the reference scenarios, resulting in time savings from 36% to more than 80%.

Table 2: Performance comparison for *App-2*.

Num. nodes	**Gap to optimum**	**Relative savings**	**Absolute savings**
5	3.6%	49.7%	465 s
10	6.0%	45.0%	341 s
15	2.4%	40.2%	275 s
20	0.7%	40.0%	261 s
25	1.5%	36.3%	247 s
30	0.0%	39.4%	235 s
35	0.7%	35.9%	193 s
40	1.2%	71.9%	913 s
45	0.8%	53.1%	397 s
50	2.8%	48.8%	369 s

In addition, the Shapiro-Wilk test has been done to prove that, for each application considered (App-1, App-2 and App-6) and number of nodes, the cost (execution time) fits a Normal distribution. Next, a variance-ratio test has been performed with significance level alpha=0.05 and proved that there is no sample evidence to reject the population variances equality. Finally, the population means have been compared through a test of differences among means with significance level $\alpha = 0.05$ and proved that there is no sample evidence to reject the population means equality. Therefore, based on the statistical analysis, it can be verified that the SOM provides performance results in terms of cost very

Table 3: Performance comparison for *App-6*.

Num. nodes	**Gap to optimum**	**Relative savings**	**Absolute savings**
5	0.4%	61.6%	1641 s
10	4.1%	64.9%	1430 s
15	0.4%	58.8%	1115 s
20	2.1%	61.1%	1169 s
25	7.7%	51.3%	881 s
30	9.7%	57.8%	1036 s
35	-	55.1%	862 s
40	-	80.2%	3040 s
45	-	66.4%	1425 s
50	-	66.6%	1435 s

close to OMM. Such evidence can be observed in Fig. 6, where 95% confidence intervals for the mean are depicted.

Despite the results obtained, it is also a must to analyze performance in terms of time to solve the different problem instances, since scalability of the proposed models may be compromised when increasing the number of nodes in the network. Fig. 7 represents the average time required to solve the problem instances for both models SOM and OMM. In addition, 95% confidence intervals for the mean are depicted for OMM and SOM lines. It can be seen that the time to solve the problem grows exponentially for OMM, being this growth very clear with applications having more than one task. Moreover, as previously described, some instances could not even be solved optimally in more than 2.5 hours. Notwithstanding, there is few growth with the number of nodes when SOM is considered. Indeed, the average time to solve the problem instances is below 1 second and below 2 seconds when *App-2* and *App-6* are considered, respectively. Regarding *App-1*, the time raises up to 80 seconds for 50 nodes. The reasoning behind this fact is the number of input data files in *App-1*'s tasks. This number is greater than those assigned to tasks in the other cases, this is, 4 input data files for the task in *App-1* against 2 input data files for tasks in *App-2* and *App-6*, as shown in Table 1 (Max. Files per Task). Thus, *App-1*'s combinatorics grow as the number of nodes turns to be large. This combinatorial growth is greater than that of task stages in *App-2* and *App-6*. So, when the number of nodes is considerably large, the solving time for *App-1* tends to be larger than that of the other applications.

In summary, in view of the results shown, it is clear that the solutions found by the optimization models for workload distribution dramatically reduce the applications' execution time, compared to the original programs. Although solutions provided by the OMM are optimal, the SOM has been proven to be an effective, yet efficient model to find almost optimal solutions in very short solving times. The execution times for the SOM optimized problem instances are very close to the optimal times, while the model scales very well with the

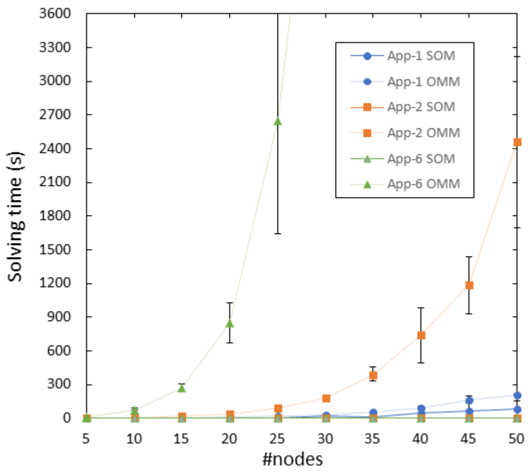

Figure 7: Solving time against number of nodes.

number of nodes and the number of tasks and stages. Assuming the short SOM solving times reported to be below 1 and 2 seconds, and considering that the absolute time savings for all optimized applications are in the order of some hundreds of seconds, the whole optimization process does not add significant overhead to the total application execution time. This means that comparing the optimizing time plus the execution time against the original benchmark, time savings are still similar to those previously described, specifically, as high as 80%. For this reason, the authors believe that the proposed SOM strategy can be utilized to efficiently and effectively distribute workloads aimed at reducing applications' execution time in global heterogeneous systems.

7. Conclusions

The research work presented in this paper is part of a global resource management framework designed to manage large volumes of heterogeneous computing devices distributed in some type of intelligent environment. This paper presents the technologies used in the module responsible for finding the best mapping between an application instance launched for execution and the availability of resources, for optimal performance. The novelty of this approach is that the tasks to resources allocation is aware of the data flow, considering both static data sources and run-time data between running tasks, and providing an optimal solution to a highly complex problem.

Two mathematical models of mixed integer linear programming have been designed to minimize the execution time of an application, treating the problem globally (Optimal Matching Model) and by levels (Stage Optimization Model), taking into account the tasks that can be run in parallel as well as the data flow between them. Multiple experiments have been carried out to verify the efficiency and effectiveness of the proposed models, and shown that although OMM provides optimal solutions to the problem, it does not scale well, as expected. However, it allows us to verify that SOM provides performance results in terms of cost very close to the optimal solution, but feasible in terms of solving time. Specifically, results show that the SOM provides optimal solutions in most evaluated scenarios (76%), while near-optimal solutions were very close to the optimal ones, with a performance difference below 6% on average. Moreover, considering all the scenarios evaluated, the average solving time for SOM is of few seconds (below 9 seconds), while it raises to hundreds of seconds (more than 450 seconds) when OMM is considered. Given the fact that SOM scales well as the number of nodes, tasks and stages increases, it is our candidate to be applied in a real optimization module.

It is out of the scope of this work to analyze how the nodes discovery and availability is monitored. Instead, in this paper a static snapshot of the available resources set has been assumed. Addressing robustness, volatility, and mobility in this scenario is a challenging topic and will be part of future stages of this research. Furthermore, in this work, an application instance is assumed with a static and known set of tasks, which is not always a realistic assumption. A future line of research will be undoubtedly leveraging machine learning technology to forecast the application behaviour and provide accurate estimation of its performance. Finally, additional efforts will be made to evaluate heuristic solvers to find near-optimal solutions at extremely short times.

Acknowledgment

This work has been supported by the Spanish Ministry of Science, Innovation and Universities and by the European Regional Development Fund (FEDER) under contract RTI2018-094532-B-I00, and has been partially supported by Huawei Technologies, Santa Clara, CA.

References

[1] A. Botta, W. de Donato. V. Persico, A. Pescapé. *On the Integration of Cloud Computing and Internet of Things.* In 2nd International Conference on Future Internet of Things and Cloud, Barcelona, Spain, August 2014.

[2] F. Bonomi, R. Milito, J. Zhu, S. Addepalli. *Fog Computing and its Role in the Internet of Things.* In 1st Workshop on Mobile Cloud Computing, Helsinki, Finland, August 2012.

[3] Y. C. Hu, et al. *Mobile Edge Computing: A key technology towards 5G.* ETSI white paper, September 2015.

[4] J. Garcia, et al. *Do we really need cloud? Estimating the fog computing capacities in the city of Barcelona.* In Workshop on Managed Fog-to-Cloud, as part of the 11th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), Zurich, Switzerland, November 2018.

[5] IBM Research @ Insight 2015, Global Technology Outlook. *Data Transforming Industries.* https://www.sgrp.ch/images/sgrpdata/events/IBMResearch.pdf [Accessed: January 2021].

[6] M. W. Convolbo, J. Chou. *Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources.* In Journal of Supercomputing, Vol. 72(3), January 2016.

[7] Y. Hu, H. Zhou, C. de Laat, Z. Zhao. *Concurrent container scheduling on heterogeneous clusters with multi-resource constraints.* In Future Generation Computing Systems, Vol. 102, January 2020.

[8] J. Garcia, X. Masip-Bruin, Y. Lu. *The Progressive Mapping System Architecture for Global Resources Management.* In 28th IEEE International Conference on Computer Communications and Networks (ICCCN), Valencia, Spain, July 2019.

[9] L. Smarr, C. E. Catlett. *Metacomputing.* In Communications of the ACM, Vol. 35(6), June 1992.

[10] A. S. Grimshaw, W. A. Wulf. *The LegionVision of a Worldwide Virtual Computer.* In Communications of the ACM, Vol. 40(1), January 1997.

[11] G. Fedak, C. Germain, V. Neri, F. Cappello. *XtremWeb: a generic global computing system.* In First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), Brisbane, Australia, May 2001.

[12] D. P. Anderson. *BOINC: A System for Public-Resource Computing and Storage.* In 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, November 2004.

[13] F. Berman, G. Fox, A.J.G. Hey (editors). *Grid Computing: Making the Global Infrastructure a Reality.* Wiley (ISBN: 978-0-470-85319-1), April 2003.

[14] L. Wang, et al. *Scientific Cloud Computing: Early Definition and Experience.* In 10th IEEE International Conference on High Performance Computing and Communications, Dalian, China, September 2008.

[15] T. Soyata et al. *Accelerating Mobile-Cloud Computing: A Survey.* In Communication Infrastructures for Cloud Computing (chapter 8), IGI Publishing Hershey, PA, September 2013.

[16] X. Chen, L. Jiao, W. Li, X. Fu. *Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing.* In IEEE/ACM Transactions on Networking, Vol. 24(5), October 2016.

[17] J. S. Murthy. *EdgeCloud: A Distributed Management System for Resource Continuity in Edge to Cloud Computing Environment.* In the Rise of Fog Computing in the Digital Era, 2019.

[18] X. Masip-Bruin, et al. *mF2C: Towards a Coordinated Management of the IoT-fog-cloud Continuum.* In 4th Workshop on Experiences with the Design and Implementation of Smart Objects (SmartObjects2018), Los Angeles, CA, June 2018.

[19] J. Garcia, et al. *A Preliminary Model for Optimal Load Distribution in Heterogeneous Smart Environments.* In the 20th IEEE International Conference on High Performance Switching and Routing (HPRS), Xi'an, China, May 2019.

[20] A. Asensio, et al. *Designing an Efficient Clustering Strategy for Combined Fog-to-Cloud Scenarios.* Future Generation Computer Systems, Vol. 109, August 2020.

[21] X. Q. Pham, E. Huh. *Towards task scheduling in a cloud-fog computing system.* In 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa, Japan, October 2016

[22] J. Wang, D. Li. *Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing.* Sensors (Basel). Vol. 19(5), March 2019.

[23] D. Hoang, T. D. Dang. *FBRC: Optimization of task Scheduling in Fog-Based Region and Cloud*, In IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, August 2017.

[24] R. Xu, et al. *Improved Particle Swarm Optimization Based Workflow Scheduling in Cloud-Fog Environment.* In Business Process Management Workshops. Lecture Notes in Business Information Processing, Vol. 342, 2018. Springer, Cham.

[25] M. Abdel-Basset, et al. *Energy-Aware Marine Predators Algorithm for Task Scheduling in IoT-based Fog Computing Applications.* In IEEE Transactions on Industrial Informatics, June 2020.

[26] M. Abdel-Basset, D. El-Shahat, M. Elhoseny, H. Song. *Energy-Aware Metaheuristic Algorithm for Industrial Internet of Things Task Scheduling Problems in Fog Computing Applications*, in IEEE Internet of Things Journal, July 2020.

[27] M. K. Hussein, M. H. Mousa. *Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization.* In IEEE Access, Vol. 8, February 2020.

[28] L. Liu, et al. *Dependent Task Placement and Scheduling with Function Configuration in Edge Computing.* In IEEE/ACM 27th International Symposium on Quality of Service (IWQoS), Phoenix, AZ, June 2019.

[29] D. Zeng, et al. *Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System.* In IEEE Transactions on Computers, Vol. 65(12), December 2016.

[30] C. Barros, V. Rocio, A. Sousa, H. Paredes. *Job Scheduling in Fog Paradigm - A Proposal of Context-aware Task Scheduling Algorithms*, In International Conference on Information Technology Systems and Innovation (ICITSI), Bandung - Padang, Indonesia, 2020.

[31] M.-Q. Tran, et al. *Task Placement on Fog Computing Made Efficient for IoT Application Provision*, Wireless Communications and Mobile Computing, Vol. 2019, January 2019.

[32] S. Venugopalan, O. Sinnen. *ILP Formulations for Optimal Task Scheduling with Communication Delays on Parallel Systems.* In IEEE Transactions on Parallel and Distributed Systems, Vol. 26(1), January 2015.

[33] C. Valouxis, et al. *DAG Scheduling using Integer Programming in heterogeneous parallel execution environments.* In 6th Multidisciplinary International Scheduling Conference (MISTA), Ghent, Belgium, August 2013.

[34] S. Sengupta, J. Garcia, X. Masip-Bruin, A. Prieto. *An Architectural Schema for Performance Prediction Using Machine Learning in the Fog-to-Cloud Paradigm.* In the 10th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, USA, October 2019.

[35] Y. Zhang, et al. *Cost Efficient Scheduling for Delay-Sensitive Tasks in Edge Computing System.* In IEEE International Conference on Services Computing (SCC), San Francisco, CA, 2018.

[36] K. H. Rosen (editor). *Handbook of Discrete and Combinatorial Mathematics*, CRC Press ISBN 0-8493-0149-1 (2000).

[37] *SageMath, the Sage Mathematics Software System (Version 9.0)*, The Sage Developers, 2020, https://www.sagemath.org [Accessed: January 2021].

[38] IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio [Accessed: January 2021].