IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 6, 2018, accepted August 12, 2018, date of publication August 29, 2018, date of current version September 21, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2867683

Optimal Task Scheduling for Distributed Cluster With Active Storage Devices and Accelerated Nodes

MOHAMMED S. BENSALEH[1], (Senior Member, IEEE), YAMAN SHARAF-DABBAGH[2], HAZEM HAJJ[2], (Senior Member, IEEE), MAZEN A. R. SAGHIR[2], (Senior Member, IEEE), HAITHAM AKKARY[2], HASSAN ARTAIL[2], (Senior Member, IEEE), ABDULFATTAH M. OBEID[1], (Senior Member, IEEE), AND SYED MANZOOR QASIM[1], (Senior Member, IEEE)

[1]Communications and Information Technology Research Institute, King Abdulaziz City for Science and Technology, Riyadh 11442, Saudi Arabia
[2]Department of Electrical and Computer Engineering, American University of Beirut, Beirut 11072020, Lebanon

Corresponding author: Syed Manzoor Qasim (mqasim@kacst.edu.sa)

This work was supported in part by the King Abdulaziz City for Science and Technology and in part by the Masri Institute at American University of Beirut.

ABSTRACT With advancements in compute-intensive and memory-bound applications, the need for faster and more energy-efficient processing platforms continues. In support of these advancements, heterogeneous platforms have been proposed to enhance the performance and efficiency in the cloud. These platforms include field programmable gate arrays and graphical processing units in addition to general-purpose processors. Furthermore, there is a strong interest in advancing active solid-state drives to support both storage and computation. In this paper, we present a generic formulation to support the modeling of such a heterogeneous cloud environment, without being specific to a particular cloud platform such as Spark or Hadoop. We represent the cloud as a collection of clients, middleware control nodes, and high performance compute nodes (HPN), where the HPNs represent the options of advanced compute technologies in a heterogeneous cloud. The objective of the paper is to present a simple and efficient formulation for scheduling applications in such a heterogeneous cloud. Consistent with recent software modeling of artificial intelligence applications, we propose to map applications into directed acyclic graph representations of tasks. The optimization problem is then formulated to infer the best scheduling of tasks on the HPNs, while minimizing the overall execution time and data communication delays between nodes. Unlike existing scheduling algorithms that assume equal performance across nodes, our formulation explicitly takes into account the different compute capabilities of the heterogeneous nodes. The resulting task scheduling is then evaluated to provide insights into the performance gains with the proposed advanced heterogeneous cloud computing environment. The results show improved performance when comparing the proposed task-scheduling algorithm with the genetic algorithm and heterogeneous earliest finish time algorithms. We also show the performance gains achieved with the optimal task scheduling on a heterogeneous cloud system as compared with a conventional CPU-only cloud system.

INDEX TERMS Distributed architectures, heterogeneous cloud system, optimization, reconfigurable hardware, scheduling and task partitioning.

I. INTRODUCTION

The ever-increasing demand for more processing capabilities is driving cloud service providers, like Google, Microsoft, and Amazon, to expand their datacenter infrastructure. As a result, datacenters' power consumption by a large extent, and network overheads continue to rise in proportion to the number of nodes in these systems. An alternative solution is to leverage special purpose hardware accelerators to provide additional computational power to individual cluster nodes. In addition to General-Purpose Processors (GPP), heterogeneous platforms have been proposed to enhance performance and efficiency in the cloud. Such heterogeneous systems are

typically comprised of conventional cores (CPUs) and unconventional cores such as Graphical Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs). Such a heterogeneous computing system employs both conventional and unconventional cores on different types of tasks, which brings improved performance and power efficiency through specialization. Furthermore, and recently, Intel unveiled a prototype for an FPGA-accelerated storage technology [1]. The flexibility and variations in such advanced compute node performance necessitate a simple method for modeling the efficient scheduling of applications in such an environment, which is the subject of this paper.

Earlier work [2], [3] has showed that data-centric [4] and High Performance Computing (HPC) [5] applications can benefit from different type of compute cores in order to achieve a computational intensive goal. The benefits translate into an improvement in overall power efficiency and performance of a system. Several major examples of such gains can be found in the ever-growing list of heterogeneous systems on the green500 list [6]. In the last decade, scalable and efficient frameworks have been developed to address the continually growing needs in big data computing systems. Most common systems include Apache Hadoop [7] and Spark [8], [9]. While these frameworks already comprehend many scheduling solutions that take many loading and demand considerations into account, opportunities continue to exist in retrofitting latest research advances that consider future technologies.

As an example, and with the emerging trend of accelerator-enabled datacenters, the research community is witnessing extensive research to integrate and efficiently deploy special-purpose hardware accelerators (GPUs and FPGAs) into these data computing frameworks. Research efforts in [10]–[14] target GPU integration within the Spark framework. In [15]–[17], FPGA integration with SPARK and Hadoop are investigated. The target of the integration is to provide a set of programming APIs for big data processing applications to easily utilize those accelerators. However, these solutions have not taken full advantage of the emerging technologies, including the use of active storage systems.

The main question that rises for such heterogeneous systems is how to efficiently schedule the application tasks. To answer this question, researchers have typically examined a two-step process, which involves prioritization of tasks then processor selection, while considering the specific applications and applicable environment constraints. Thakur *et al.* [18] consider dynamic voltage scaling for optimized energy consumption from different computing needs. Xu *et al.* [19] and Shojafar *et al.* [20] consider custom genetic algorithm with different priority queues for different sub-tasks.

In this paper, we focus on shortest execution time while considering different performance options of the heterogeneous compute nodes. Given an application of n tasks and a system of m heterogeneous computational nodes, what is the assignment of each of the n tasks to each of the m nodes to achieve best performance? We provide a generic representation of the distributed heterogeneous system as consisting of three layers: application layer, middleware servers' layer, and High Performance Nodes (HPNs) layer. Fig. 1 shows an example of heterogeneous cluster with one client, three middleware servers, and five HPNs.

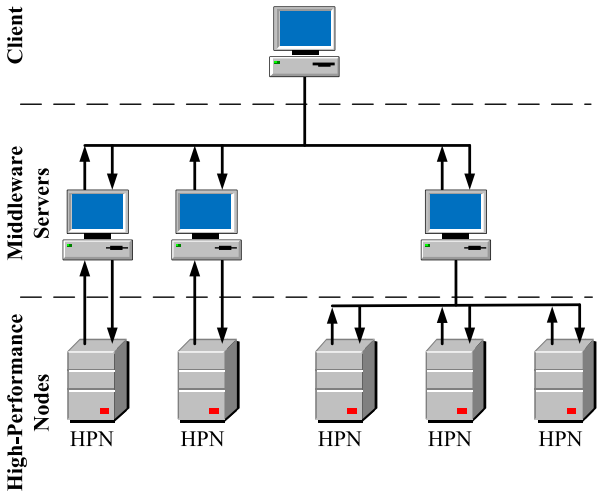

FIGURE 1. System architecture of heterogeneous cluster.

The application layer or client layer initiates the user communication with the system for data processing. The middleware server layer receives the requests from the application layer and acts as a management middle stage between the application and HPN layers. This layer manages operation of HPNs such as scheduling and task allocation. It also handles control of processing with communication and aggregation operations. Depending on the size of the cluster, the middle layer may consist of two primary nodes as the case with small clusters managed in a Spark environment. The HPN layer includes the core heterogeneous components of the system with specialized high performance nodes. A HPN consists of a specialized hardware (FPGA, GPU, or active SSD) connected to memory storage drives.

In its most general form, the problem of scheduling tasks of a graph onto a set of different machines is an NP-complete problem [21]. Therefore, optimal task assignments are only tractable for relatively small problems, whereas only heuristic solutions are feasible for large problems. As a result, over several years, a number of heuristics have been developed that attempt to strike a good balance between running time, complexity and schedule quality [22]. A number of heuristics suitable for task scheduling on heterogeneous resources have been suggested; a comparative evaluation of twenty different heuristics can be found in [23]. Among these heuristics, the Heterogeneous Earliest Finish Time (HEFT) heuristic [24] has become a good baseline for comparison, having the advantage of simplicity and producing generally good schedules with a short makespan [25]. The Genetic Algorithm (GA) has also been widely reckoned as useful meta-heuristics for obtaining high quality solutions for

a broad range of combinatorial optimization problems including the task scheduling problem [26], [27]. The custom HEFT and the GA are implemented in this paper to solve the task scheduling problem in a heterogeneous and distributed cluster.

The main contributions of this paper are as follows:

- A generic abstraction and formulation of the heterogeneous cluster-scheduling problem is presented. It is formulated as an optimization problem to map application tasks to computational nodes in a heterogeneous cluster.
- Optimal allocation solutions for small-scale clusters and suboptimal solutions of the algorithm with large size clusters are provided. In the latter case, a custom GA and HEFT heuristic solutions are used to adapt to the proposed formulation.
- The proposed algorithm is applied on both, a conventional CPU-only cluster and the heterogeneous system, and the merits of the heterogeneous cluster as compared with the conventional system are analyzed.

The rest of the paper is organized as follows: Section II presents the related work. Section III describes the details of the generic heterogeneous cluster topology and scheduling model. In Section IV, we present the proposed optimal allocation algorithm. Custom heuristic algorithms are developed and used for comparison in Section V. Simulation results are presented and analyzed in Section VI. Finally, conclusion is presented in Section VII.

II. RELATED WORK

Scheduling in distributed systems can be classified into different categories as described in [28]. The highest level of classification is whether the allocation specifies how the tasks are executed locally on each computing node or how the tasks are executed globally on the whole distributed system. The second level of categorization under the global scheduling is the choice between dynamic and static scheduling. In dynamic scheduling, task allocation decisions are made in real-time as the application executes. In static scheduling, decisions are made a priori, before running the application. Since scheduling problems are NP-hard, finding an optimal allocation of tasks is not always possible. Therefore, the third level in the hierarchy is the choice between optimal or suboptimal scheduling algorithms. For both the optimal and suboptimal scheduling, the goal is to achieve the best performance, minimum resource consumption, best reliability, or a combination of the previous goals. Fig. 2 shows the taxonomy presented before. This research work falls under the category of global-static scheduling. We propose an optimal algorithm to achieve the best performance along with the implementation of two heuristics for performance maximization.

Wang *et al.* [29] studied task allocation on heterogeneous distributed systems to minimize the total energy consumption. The authors formulated the problem as an integer linear programing and used heuristic algorithms to solve it. However, the formulation proposed in [29] did not consider the case of having specialized accelerator nodes.

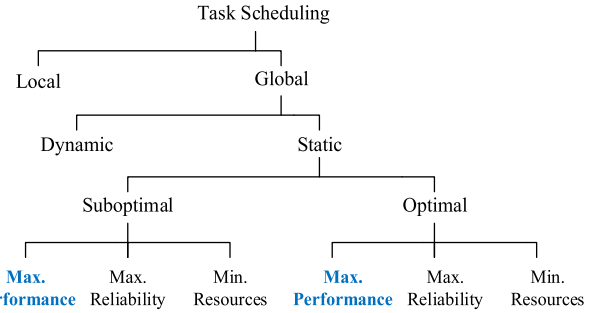

FIGURE 2. The main classifications of task scheduling algorithms in distributed systems. The maximum performance algorithms in both the optimal and suboptimal case are the focus of this work.

Other research work, such as in [30] focus on optimal scheduling of tasks to reduce execution time. Visalakshi and Sivanandam [30], the communication links were considered identical and used the meta-heuristic Hybrid Particle Swarm Optimization (HPSO) to solve the formulation. Xu *et al.* [31] minimize the three goals of scheduling in one objective function. They used the Chemical Reaction Optimization (CRO) as a solving heuristic for the formulation. In [32], reliability and performance were maximized, and Ant Colony Optimization (ACO) was used in this work. Benoit *et al.* [33] focused on improving the performance of running group of tasks instead of individual tasks. The limitation in their work is that each node can process at most one group of tasks. In [34], the objective functions were formulated to minimize both the execution time of the application along with minimizing the workflow of the application.

Heuristic approaches were presented in [35]–[44]. In [35], fast ordinal optimization method is introduced to find suboptimal task allocation schedule in a short timeframe. Yadav *et al.* [36] proposed a greedy heuristic scheduling algorithm. The heuristic allocates the tasks with the heaviest communication links on the fastest link available. Ucar *et al.* [37] implemented six heuristic algorithms to solve the task assignment problem including the Genetic Algorithm (GA) implemented in this work, but they assumed all the communication links are identical. The GA algorithm performed the best among all algorithms. Kang *et al.* [38] proposed an iterative greedy algorithm to allocate tasks to enhance performance. Attiya and Hamam [39] implemented the Simulated Annealing (SA) heuristic algorithm to find out the best performance of the system while taking into consideration system constraints. In [40] and [41], they clustered the tasks into different groups where each group is executed on a computing node of the parallel system. In [42]–[44], graph mining techniques were used to come up with the best task allocation.

As discussed, the task allocation problems have indeed been studied in the literature relying on static and dynamic optimization techniques. The differences are often in the objective functions, or the imposed constraints. Some of the closest work to ours is the work in [45] and [46]. Wen *et al.* [45] presented a method to allocate applications

into multiple cloud processing systems to achieve high reliability by choosing the most reliable cloud services available. While the concept of different clouds is somehow similar to having different types of processing units, the authors had many restrictions and simplifications in their approach that makes the problem different than ours.

First, the authors assume the workflow of the application is completely parallelizable which limits the applicability of the solution to such cases and makes distributing the application into multiple cloud services simple. For example, adding vectors is a parallelizable operation, however serial operations where the output of one operation is required for another operation is not fully parallel and requires sequential steps. In their work, the formulation to assign tasks to the systems would be to minimize the total cost without the need to map the workflow sequence of the application tasks into the constraints. For a completely parallel application, each task can be assigned to any processing unit in the system without any constraints that reflect dependencies for certain tasks. On the other hand, we account for applications that include branching and that need to be mapped as a workflow highlighting the serial parts of the application and the parallel part of an application. Such a workflow is included in our formulation as shown in the constraints of our objective function. Second, the authors assumed that the cloud services had a fully connected topology, which limits the choices of applications.

Examples of systems that are not fully connected include any processing system with star, ring, or bus architectures. In all of these examples, not all processing units are connected to each other with a direct link. In our case and as shown in Fig. 1 in this paper, the sample processing system includes nodes that are not directly connected to each other. Third, the fully connected topology simplifies the problem and eliminates the need to add constraints that map the physical topology of the processing system. In our work, we add a set of constraints to map the physical system with high processing units along with standard processing units.

Zhu *et al.* [46] proposed an optimization formulation for computing nodes used in automobiles. Their approach to build the formulation has some similarity to our work, however the key elements of our work is the derivation of the constraints applicable to heterogeneous systems. Zhu *et al.* [46] presented a solution to allocate different types of tasks with different priorities into processing units. They used assignment variables for each task and mapped the physical system architecture of processing units into the constraints of the assignment problem. Even though the authors state that they assume heterogeneous compute resources, the constraints specified in the paper did not account for multiple types of processing units' performances. Instead, the assignment of tasks was formulated as an assignment of a task to a processing unit regardless of its type (as shown in sections 3.2 and 3.3.1 of their paper [46]).

In our approach, we assign the tasks to units only if the compute node constraints are satisfied. In particular, the constraints check whether consecutive tasks should be assigned to high performance nodes connected to the same middleware or they need to be assigned to high performance nodes connected to different middlewares. As an example, tasks on high performance nodes connected with the same middleware run parallel tasks faster than high performance nodes connected to different middleware layers. The assignment constraints in our multilayered (shown in Fig. 1 and comprising the client, middleware, and compute nodes) system and that consider the heterogeneous resources are presented in section IV-C of our paper.

It can be seen from the listed constraints that accounting for multiple types of processing units increases the difficulty of the problem and requires a solution that optimizes the distribution of tasks to the multi-type processing unit systems. In addition to the formulation differences, our proposed system is expected to lead better application performance compared to solution based on formulation in [46] when task priority is not considered. For example, in our multilayer system, the work of [46] would consider all the nodes to be the same and assign the tasks on all the nodes without taking into consideration the scenario where consecutive tasks can run faster on a high performance node compared to a standard node in the middleware layer.

III. SCHEDULING MODEL

The objective of this work is to determine an optimal distribution of tasks over a distributed cloud computing environment with specialized compute nodes capable of hardware acceleration. Towards deriving the optimization formulation, the problem can be described as shown in Fig. 3. Given a set of specified inputs, the algorithm tries to determine where tasks should be allocated to achieve the best overall performance. The inputs of the optimization algorithm are the heterogeneous system topology, Data location, and Directed Acyclic Graphs (DAG). The constraints introduced in Section IV (A) exploit the data locality in the heterogeneous system by assigning smaller execution costs when tasks are allocated to the HPN where the data is stored. The decrease in cost is due to data having the processing closer to the data storage. The formulation explores all the different allocation scenarios, and weights each allocation decision differently. Assigning weights depends on how convenient are the computing nodes to execute the tasks, and depends on the distance between the computing nodes and the data locations.

A. HETEROGENEOUS SYSTEM TOPOLOGY

The heterogeneous system topology includes information about the number of physical nodes, and the network topology, which is the arrangement of physical nodes and how the data flows between these physical nodes. The description also includes the types of physical nodes: (Client (CL), MiddleWare (MW), and special high performance compute nodes (HPN). Fig. 3 shows the heterogeneous system topology for a system with five HPN nodes and three Middleware servers. Fig. 3 also shows the possible seven links in the system:

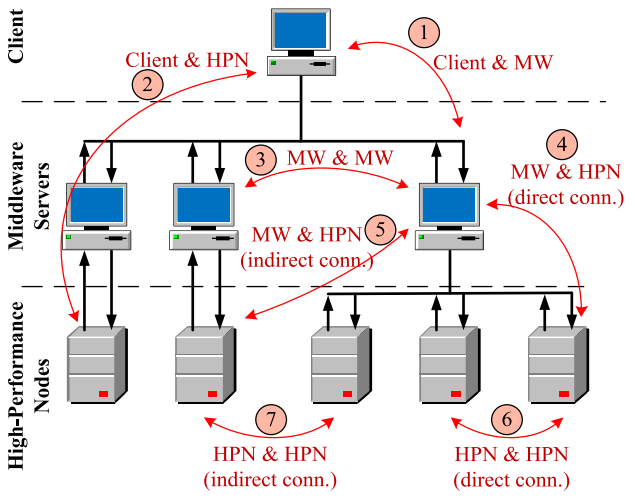

FIGURE 3. Seven possible links in a heterogeneous cluster.

1. CL—MW - Between client and middleware: when one of the tasks is allocated on a middleware node and the other is on the client.
2. CL—HPN - Between client and HPN node: when one of the tasks is allocated on a HPN node and the other is on the client. Note that in such a case, the communication cost needs to account for passing through middleware.
3. MW—MW - Between two middleware servers: when the two tasks are allocated on different middleware nodes.
4. MW—HPN (dir.) - Between middleware server and HPN node with direct connection: when one of the tasks is allocated on a HPN node and the other on a middleware node, and both are on the same network.
5. MW—HPN (indir.) - Between middleware and HPN node with indirect connection: when one of the tasks is allocated on a HPN node and the other on a middleware node, and both nodes are on different networks.
6. HPN—HPN (dir.) - Between two HPN nodes with direct connection: when the two tasks are allocated on different HPN nodes but the nodes are on the same network (connected to the same middleware).
7. HPN—HPN (indir.) - Between two HPN nodes with indirect connection: when the two tasks are allocated on different HPN nodes, and the nodes are on different networks (connected to the two middleware servers).

B. APPLICATIONS REPRESENTED THROUGH DIRECTED ACYCLIC GRAPHS (DAG)

DAGs are used to represent applications composed of tasks. A task is a set of sequential instructions to be executed on a single processing node. The size of the task is application specific and depends on the developer's choice of dividing the application into blocks. Each vertex in the DAG represents a task, and the weight of the vertex represents the execution cost of running the task on a computing node. Each edge in the DAG has a weight representing the communication cost between the two vertices (tasks) connected by the edge. As a result, changing the size of the data transferred between the tasks changes the cost of communication. Different DAGs may exist for the same application for each data size value. The characteristics of the communication signals between tasks, like network protocol, bandwidth, burst transmission, or continuous transmission, are reflected in our formulation through the weights of the edges. To obtain execution times for each task in an application, multiple profiling tools can be used, such as [47] and [48]. Using profiling tools, estimates on execution times for each task are obtained which represent the weights of the vertices in the DAG.

Old approaches used matrix representation of tasks in the application [49]. In this work, DAG will have a single entry node and single exit node to represent the start and end times of an application. An entry node is a node with no parent. An exit node is a node with no child. We make the following assumptions about the DAG:

- Since some applications may need multiple entry points reflecting multiple starting threads, a zero weight entry node is added to the DAG to provide the overall entry node. This node becomes the parent of all original entry nodes. Similarly, a zero weight exit node can be added to support multiple exit points. The exit node becomes the common child to all original exit nodes.
- The DAG includes information about Regular processing nodes, and HPN nodes capable of high processing acceleration. The weights associated with HPN nodes are lower to reflect accelerated processing and faster execution times.

C. DATA LOCALIZATION

Data localization is reflected within the DAG by adding vertices corresponding to the locations where the data reads and writes will be initiated. For each HPN node, an additional vertex is added to reflect the data read or write task for that node. The weights of the vertices for the read and write tasks are adjusted to reflect location of the data.

IV. SCHEDULING ALGORITHM

The objective of the optimization algorithm is to map the application tasks to the heterogeneous system physical nodes. We formulate the optimization problem in the framework of Mixed Integer Linear Programming (MILP), where the inputs, DAG and system topology, are represented with variables and parameters. From these variables and parameters, an objective function is formulated which characterizes the optimal solution. This objective function is constrained by a set of constraints that assure the optimal solution is feasible.

The standard MILP form is:

$$\min f(x)$$

Subject to:

$$\begin{aligned} Ax &\leq b \\ A_{eq}x &= b_{eq} \\ x &\geq 0, \end{aligned}$$

where, $x = (x_1, \ldots, x_n)$ is the vector of decision variables, A and A_{eq} are constant matrices with dimensions $m_1 \times n$ and $m_2 \times n$ respectively where m_1 is the number of inequality constraints and m_2 is the number of equality constraints. b and b_{eq} are constant vectors of sizes $m_1 \times 1$ and $m_2 \times 1$ respectively.

The values of constant vectors and matrices c, A, A_{eq}, b and b_{eq} define the exact problem at hand by reflecting the specifications of the heterogeneous system and the DAG into the optimization problem.

A. VARIABLES AND PARAMETERS

The following variables and parameters are used in the optimization problem:

1) INPUT PARAMETERS

The input parameters that need to be provided for the algorithm are:

DAG Representation for application tasks:

- T Set of all the computational **T**asks in the DAG. A task will be represented by $i \in T$. Each task corresponds to a node in the DAG
- $L_{i,j}$ **L**ink or edge between tasks $(i, j) \in T$

$$L_{i,j} = \begin{cases} 1, & \text{if there is a link from task } i \in T \text{ to task } j \in T \text{ in the DAG} \\ 0, & \text{otherwise} \end{cases}$$

- $P_{i,j}$ Indicator of task dependency between two tasks i and j. $(i, j) \in T$

$$P_{i,j} = \begin{cases} 1, & \text{if task } i \in T \text{ precedes task } j \in T \text{ in the DAG} \\ 0, & \text{otherwise} \end{cases}$$

Mapping of DAG tasks onto the heterogeneous nodes:

- $e_{i,a}$ **E**xecution time of task $i \in T$ when executed on node $a \in N$.
- $c_{i,j,a,b}$ **C**ommunication time between tasks $(i, j) \in T$ when the tasks are allocated to nodes $(a, b) \in N$ respectively. Note that this depends on $C_{i,j}$ the communication load between i, j, and the physical communication between the two nodes (a, b).

Heterogeneous System representation of physical nodes:

- N Set of all the physical Nodes in the system, each node is a HPN, a middleware, or a client node. Note that these are different from the DAG nodes.
- CL Label for a **CL**ient node in the system.
- MW Set of (labels for) all **M**iddle **W** are servers in the system $MW \subset N$.
- HPN_m Set of (labels for) **H**igh **P**erformance physical **N**odes in the system connected to a middleware node $m \in MW$.

2) SUPPORTING DECISION VARIABLES

The supporting decision variables are variables in the optimization problem. The values of the supporting decision variables are based on the values of the output decision variables of the problem.

- $A_{i,j,a,b}$ A flag to indicate the mapping of communication between tasks to the actual communication between the mapped physical nodes in the system. The value is binary.

$$A_{i,j,a,b} = \begin{cases} 1, & \text{if com. sig. between tasks } (i, j) \in T \text{ is allocated to the link between nodes } (a, b) \in N \\ 0, & \text{otherwise} \end{cases}$$

- $O_{i,j}$ A flag indicating whether two tasks overlap. The value is binary.

$$O_{i,j} = \begin{cases} 1, & \text{if task } i \in T \text{ starts before task } j \in T \text{ finish} \\ 0, & \text{otherwise} \end{cases}$$

3) OUTPUT DECISION VARIABLES

The output decision variables are the variables of the optimization problem that we are trying to find.

- $A_{i,a}$ Allocation or Assignment variable of tasks to physical nodes. The value is binary.

$$A_{i,a} = \begin{cases} 1, & \text{if task } i \in T \text{ is allocated to node } a \in N \\ 0, & \text{otherwise} \end{cases}$$

- S_i Computed Start time of task $i \in T$ relative to the start time of the entry (first) task in the DAG for a given set of tasks to nodes assignments ($A_{i,j}$, $A_{i,j,a,b}$, and $O_{i,j}$). The value is an integer representing units of time. $S_1 = 0$ the first task to execute in the DAG on any node.
- X An artificial variable to smooth the min-max problem into min X and added constraints problem. This variable represents the makespan of the application, which equals the total execution time of the application, and indicates the finish time of the exit task in the DAG. This is an integer value.

B. OBJECTIVE FUNCTION

The objective function characterizes the optimal solution. The optimal solution ensures executing the given application represented as a DAG on the given heterogeneous system as fast as possible or with the least latency possible. Therefore, the objective function:

$$\min_{Solution\, q} \max_{task\, i} (finish\_time_i), \tag{1}$$

where: $finish\_time_i$ is the finish time of task i which equals the sum of S_i, the start time for task i and $e_{i,a} \times A_{i,a}$, the execution time of task i.

The min-max problem can be reformulated as a smooth minimization problem using an artificial variable we call it X, which also represents the makespan for the application.

$$\min_{Solution\ q} X \tag{2}$$

Subject to:

$$S_i + e_{i,a} \times A_{i,a} \leq X \quad \forall i \in T \tag{3}$$

The vector of variables, associated with each scenario of possible tasks to nodes allocations, can be represented by the vector of n variables:

$$\left[A_{i,a}, A_{i,j,a,b}, X, S_i, O_{i,j}\right]$$

The size of the vector (n) is dependent on the number of variables for allocations of tasks to nodes, the size of each variable is:

- $A_{i,a}$: number of tasks in DAG $\times$ number of system nodes.
- $A_{i,j,a,b}$: number of edges in DAG $\times$ number of possible types of physical links.

where, the number of possible types of physical links $= 7$, for the heterogeneous system.

- X: scalar
- S_i: number of tasks in DAG - 1
- $O_{i,j}$: number of parallel tasks in the DAG (two parallel tasks are counted as 1).

C. CONSTRAINTS

The first constraint is the smoothing constraint already mentioned before:

$$S_i + e_{i,a} \times A_{i,a} \leq X \quad \forall i \in T \tag{4}$$

The starting time of each task S_i depends on two factors:

1. The time to execute tasks preceding it. For each edge in the DAG, the start time for the task at the head (subsequent task) of the edge is larger or equal to the sum of the start time of the task at the tail (preceding task) of the edge and the time to execute the task at the tail of the edge and the time of communication between the two nodes.
2. The time to execute tasks which are assigned on the same node as task i. We are assuming that a given node executes tasks sequentially. So task i has to wait for other tasks to finish if they were already started before task i.

To capture the constraint in the first factor, we check for each pair of successive tasks that the start time of the later task is larger than the finish time for the former task plus the time to communicate signals between them:

$$S_j \geq S_i + e_{i,a} \times A_{i,a} + c_{i,j,a,b} \times A_{i,j,a,b} \quad \forall (i,j) \in T | L_{i,j} = 1 \tag{5}$$

To include the second factor, we need to check for each pair of tasks executing on the same physical node, and check that the start time of one of these tasks is larger or equal to the finish time of the other task.

$$S_j \geq S_i + e_{i,a} \times A_{i,a} \vee S_i \geq S_j + e_{j,a} \times A_{j,a} \quad \forall a \in N, \forall (i,j) \in T | P_{i,j} = 0, A_{i,a} = A_{j,a} \tag{6}$$

This "OR" conditional constraint can be alternatively represented using the big "M" notation. Therefore, the equivalent for the previous conditional constraint is:

$$S_j - e_{i,a} \times A_{i,a} - S_i \geq -M \times O_{i,j} \tag{7}$$

$$S_j - e_{i,a} \times A_{i,a} - S_i < M \times \left(1 - O_{i,j}\right) \quad \forall (i,j) \in T | P_{i,j} = 0 \tag{8}$$

The two tasks i, j overlap when the values of $O_{i,j} = O_{j,i} = 1$. To ensure tasks run sequentially (not overlap) on a node, the tasks must not be allocated to execute on the same node at the same time. We need to ensure $\left(O_{i,j} \neq O_{j,i}\right)$. Therefore, we add the following constraint:

$$A_{i,a} + A_{j,a} + O_{i,j} + O_{j,i} \leq 3 \quad \forall (i,j) \in T, \quad \forall a \in N | P_{i,j} = 0 \tag{9}$$

An important constraint is to make sure all the tasks are allocated, and each task is allocated to one physical node only:

$$\sum_{a \in N} A_{i,a} = 1 \quad \forall i \in T \tag{10}$$

To account for hardware topology, we add the following set of constraints:

For each two consecutive tasks (i,j) in a serial route, there is one mapping for an edge to the physical communication between two nodes:

$$\sum_{(a,b) \in N} A_{i,j,a,b} \leq 1 \quad \forall (i,j) \in T, \ \forall (a,b) \in N | L_{i,j} = 1 \tag{11}$$

If the two consecutive tasks (i,j) are allocated to the same node (but not executing at same time), the communication between them is set to zero (all $A_{i,j,a,b}$ are zeros, including $A_{i,j,a,a}$):

$$2 - A_{i,a} - A_{j,a} \geq \sum_{(a,b) \in N} A_{i,j,a,b} \quad \forall (i,j) \in T, \forall (a,b) \in N | L_{i,j} = 1 \tag{12}$$

If the two consecutive tasks (i,j) are allocated to *two different HPN* nodes a and b, then the allocation variable $A_{i,j,a,b}$ must be equal to 1 indicating they are physically connected. Furthermore, for the case, where both these nodes are connected to the *same middleware server*, we can check over nodes connected as such, where (a,b) belong to the set of HPN nodes connected to the same middleware:

$$\sum_{a \in HPN_m} A_{i,a} + \sum_{b \in HPN_m} A_{j,b} - 1 \leq A_{i,j,a,b} \quad \forall (i,j) \in T, \forall m \in MW, a \in HPN_m, b \in HPN_m \left| L_{i,j} = 1 \right. \tag{13}$$

If the two consecutive tasks (i, j) are allocated to *two different HPN* nodes a and b, then the allocation variable $A_{i,j,a,b}$ must be equal to 1 indicating they are physically connected. This case is different than the previous case in, that nodes a and b are *connected to different middleware servers*. Where a belong to the set of HPN nodes connected to a middleware m, and b belong to the set of HPN nodes connected to middleware n, where $m \neq n$:

$$\sum_{a \in HPN_m} A_{i,a} + \sum_{(n \in MW | n \neq m)} \sum_{b \in HPN_n} A_{j,b} - 1 \leq A_{i,j,a,b} \quad (14)$$

$$\sum_{(n \in MW | n \neq m)} \sum_{a \in HPN_n} A_{i,a} + \sum_{b \in HPN_m} A_{j,b} - 1 \leq A_{i,j,a,b}$$
$$\forall (i, j) \in T, \quad \forall (m, n) \in MW,$$
$$a \in HPN_m, \quad b \in HPN_n | n \neq m \quad (15)$$

If the one of the consecutive tasks (i, j) is allocated to an *HPN node* and the other task is allocated to a *middleware server*, then the allocation variable $A_{i,j,a,b}$ must be equal to 1 indicating they are physically connected. Furthermore, for the case where a belong to the set of HPN nodes connected to the middleware b:

$$\sum_{a \in HPN_m} A_{i,a} + A_{j,m} - 1 \leq A_{i,j,a,m} \quad (16)$$

$$A_{i,m} + \sum_{a \in HPN_m} A_{j,a} - 1 \leq A_{i,j,m,a}$$
$$\forall (i, j) \in T, \quad \forall m \in MW, \ a \in HPN_m \, | L_{i,j} = 1 \quad (17)$$

If the one of the consecutive tasks (i, j) is allocated to an *HPN node* and the other task is allocated to a *middleware server*, then the allocation variable $A_{i,j,a,b}$ must be equal to 1 indicating they are physically connected Furthermore, for the case where a belong to the set of HPN nodes connected to a middleware m other than the middleware b, $(m \neq b)$:

$$\sum_{a \in HPN_m} A_{i,a} + \sum_{n \in MW | n \neq m} A_{j,n} - 1 \leq A_{i,j,a,n} \quad (18)$$

$$\sum_{n \in MW | n \neq m} A_{i,n} + \sum_{a \in HPN_m} A_{j,a} - 1 \leq A_{i,j,n,a}$$
$$\forall (i, j) \in T, \quad \forall (m, n) \in MW, \ a \in HPN_m$$
$$| n \neq m, L_{i,j} = 1 \quad (19)$$

If the two consecutive tasks (i, j) are allocated to *two different middleware servers* a and b, then the allocation variable $A_{i,j,a,b}$ must be equal to 1, indicating they are physically connected:

$$\sum_{m \in MW} A_{i,m} + \sum_{n \in MW} A_{j,n} - 1 \leq A_{i,j,m,n}$$
$$\forall (i, j) \in T, \quad \forall (m, n) \in MW \, | L_{i,j} = 1 \quad (20)$$

If the one of the consecutive tasks (i, j) is allocated to a *middleware server* and the other task is allocated to a *client node*, then the allocation variable $A_{i,j,a,b}$ must be equal to 1 indicating they are physically connected:

$$\sum_{m \in MW} A_{i,m} + A_{j,a} - 1 \leq A_{i,j,m,a} \quad (21)$$

$$A_{i,a} + \sum_{m \in MW} A_{j,m} - 1 \leq A_{i,j,a,m}$$
$$\forall (i, j) \in T, \quad m \in MW \, | L_{i,j} = 1 \ , \ a = CL \quad (22)$$

If the one of the consecutive tasks (i, j) is allocated to an *HPN node* and the other task is allocated to a *client node*, then the allocation variable $A_{i,j,a,b}$ must be equal to 1 indicating they are physically connected:

$$\sum_{m \in MW} \sum_{b \in HPN_m} A_{i,b} + A_{j,a} - 1 \leq A_{i,j,b,a} \quad (23)$$

$$A_{i,a} + \sum_{m \in MW} \sum_{b \in HPN_m} A_{j,b} - 1 \leq A_{i,j,a,b} \quad \forall (i, j) \in T,$$
$$m \in MW, \quad b \in HPN_m \, | L_{i,j} = 1 \ , \ a = CL \quad (24)$$

V. HEURISTIC ALGORITHMS

Since task scheduling is an NP-hard problem [8], we resort to heuristic algorithms to find fast suboptimal solutions for large problems. The Genetic algorithm (GA) and the Heterogeneous Earliest Finish Time algorithm (HEFT) are employed to solve the optimization problem.

A. GENETIC ALGORITHM

Genetic Algorithms (GAs) have been widely used and proven to obtain high quality solutions [37]. GA searches for a solution in an initial population of solutions. During the iterative search process, new solutions are created through a mating process. A fitness function is used to measure the quality of each candidate solution. One of the biggest advantages of the GA algorithm is the scalability relative to the problem size.

1) INITIAL POPULATION AND REPRESENTATION OF SOLUTIONS

The most commonly used representation of solutions is an array of tasks for each machine [50]. For simplicity, each machine is represented by a number from $1, \ldots, N$ where N is the total number of machines in the system. Each cell in the array of tasks will have a value representing the machine number where the task will be executed on. For example, if we have a DAG with 5 tasks and a system with a total of 3 nodes, then a sample encoding of a solution would be as in Fig. 4.

Random initialization of population is usually used to generate the initial set of solutions. Heuristics are used also to generate the initial population for the GA algorithm. In this work, beside the randomly generated set of solutions, we add three possible solutions to the population, we add initial solutions where all tasks are allocated on the HPN nodes, since the system's HPN nodes are likely to have the data stored on them compared to other system nodes, such as MW nodes.

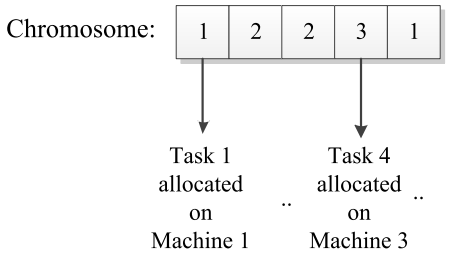

FIGURE 4. Sample chromosome in the genetic algorithm representing possible distribution of 5 tasks in a system of 3 physical nodes.

2) FITNESS FUNCTION

The fitness function of the genetic algorithm evaluates each possible solution at every iteration of the genetic algorithm. Given the solution, the allocation of tasks to nodes, the fitness function simulates the running of the application with the expected communication costs based on the given allocation.

Algorithm 1 presents the main steps of the genetic algorithm used.

Algorithm 1 Genetic Algorithm

Input: initial population of possible solutions
Output: task to node allocation
Evaluate the initial population using the fitness function
Repeat
Apply crossover and mutation functions to generate a new child generation of solutions.
Evaluate the child generation
Save the solutions from the child and parent generations with the least fitness function cost.
The saved solutions form the new parent generation.
Until the stopping criterion is reached

B. HETEROGENEOUS EARLIEST FINISH TIME ALGORITHM

In this algorithm, a task with the highest rank would be allocated to the computing node, which minimizes its finish time. The ranking for the tasks is called an upward rank. The upward rank is calculated by taking the largest sum of the computation mean cost and communication mean cost along any path from the task to an exit node. The algorithm starts with highest upward rank task and down.

VI. EVALUATION

To evaluate the performance of the proposed algorithm, we resort to a simulation framework where the proposed algorithm to allocate tasks to compute nodes is compared to the performance of GA, HEFT and the same algorithm but on a conventional (homogeneous multiprocessors) system. The following factors are considered:

1. Hardware system configurations:
   a. Number of processing nodes: takes value from 3 to 37 processing nodes.
   b. Ratio of HPN nodes to MW servers: takes value from 1 HPN for each MW server to 11 HPNs for each MW server.
   c. Acceleration speed of the HPN nodes: takes values from 1x to 20x. We changed the values from 1x to 20x to cover the average possible acceleration speeds.
2. Application complexity as represented by its DAG Size. This includes the number of tasks or nodes in the DAG, and the amount of dependency among tasks reflected in the number of edges. To reflect the size of the DAG in the experiments we varied the number of tasks from 10 to 50 tasks.

Fig. 5 shows the set of conducted simulations. The first three columns represent the different options for hardware configuration. The fourth column represents the size of the DAG. The experiments consider different combinations of the values in these different columns. To show the effect of each factor mentioned before, we vary the value of the parameter under study while other parameters take fixed values. The red bolded line represents the fixed values that each parameter would take when it is not under study. For example, in the case where we assess the ratio parameter, the value of the ratio parameter changes from 1/1 to 1/11 whereas the rest of the parameters (number of nodes, acceleration and number of tasks) are set to the values of 13, 4 and 50 respectively.

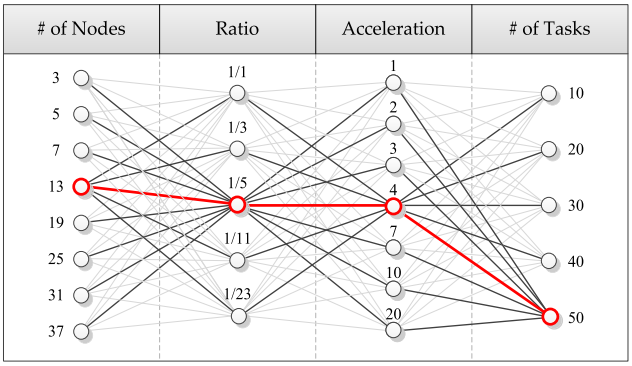

FIGURE 5. Set of conducted experiments.

A. SIMULATION SETUP

The proposed algorithm is implemented and evaluated in a simulated environment. The experiments were intended to run on DAGs representing data analytics applications, however due to their lack of availability, the experiments were performed on a set of benchmarking DAGs generated by Davidovic and Crainic [51], and commonly used for performance analysis. The DAGs provided by Davidovic feature both task execution costs and communication costs, whereas other benchmarking DAGs like the Standard Task Graph set (STG) [52] do not include communication costs in their DAGs tasks. Each test case was repeated 10 times and the average cost function value was calculated. IBM ILOG CPLEX was used as the Mixed Integer Linear

Programming (MILP) solver running on a Lenovo ThinkStation, Intel Xeon CPU E5-2620 2.00GHz, 24GB of RAM.

One important feature in the DAG is level of task dependencies among tasks, which is also reflected by the maximum parallelism [53]. The level of parallelism limits the performance of running the DAG on the system since we cannot exceed the maximum speedup that can be achieved even as the number of compute nodes are increased. The structure of the DAG and the number of parallel tasks determine how much parallelism the DAG has. To better show the effect of increasing the number of nodes in the system on overall performance, we chose the DAGs from the benchmarking dataset with the biggest parallelism factor.

The parallelism can be measured by the following equation:

$$Parallelism = Time(1)/Time(\infty) \tag{25}$$

Where, *Time(1)* represents the *work*, which is the total running time of the application when allocated to a single processing node which is the time spent for the application to run sequentially. $Time(\infty)$ is the span that consists of the time needed to run the most expensive path in the application from beginning to end; it is also called the critical path length. Fig. 6 provides a visual illustration of the parallelism in a DAG. Both DAGs in Figure 6a and Fig. 6b have the same number of tasks, but the DAG in Fig. 6a has more dependencies than the DAG in Fig. 6b. As a result, the DAG in Fig. 6a has more sequential tasks. These differences are reflected in the level of parallelism. The DAG in Fig. 6a has a parallelism factor of 5 where the DAG in Fig. 6b has a 15.4 parallelism factor. The DAG in Fig. 6b has a maximum of three sequenced tasks shown in the figure as three levels of tasks.

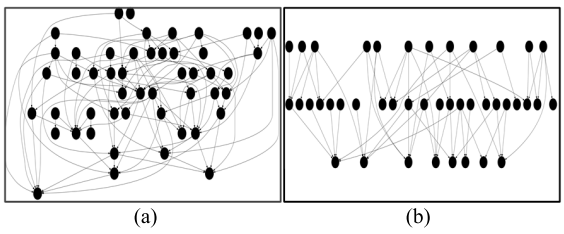

FIGURE 6. Parallelism in DAGs. (a) DAG with 50 tasks and parallelism factor of 4.979. (b) DAG with 50 tasks and parallelism factor 15.4.

B. IMPACT OF THE NUMBER OF COMPUTE NODES

In this set of experiments, we varied the number of nodes from 3 to 37, with other parameters (ratio of HPNs to MWs, acceleration and number of tasks) are set to the values of 1/5, 4 and 50 respectively. The results are shown in Fig. 7, which shows that the heterogeneous system is on average 2 to 3 times better than a conventional system without the HPNs. The improvement in performance takes an exponential curve instead of linear slope curve as nodes are increased. The exponential drop is due to the increase of communication costs as more physical nodes are used to run the application.

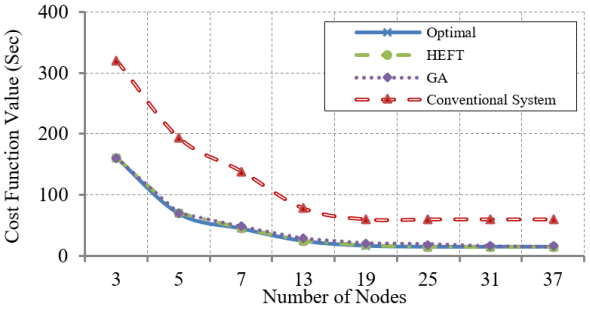

FIGURE 7. Impact of increasing the number of computing nodes.

We also notice that as the number of nodes becomes larger than 19, the performance improvements tend to saturate. This is due to the parallelism limit of the DAG. Therefore, increasing the number of nodes in the heterogeneous system enhances the performance and decreases the running time of the application, but the improvement is limited by the level of parallelism in the application.

Increasing the number of nodes in the system enhances the performance of the system up to the point that all possible parallel tasks can be handled. Additional nodes beyond that point will cause nodes to be in idle states during the run time of the application. From this set of experiments, we observe that to achieve better performance more compute nodes should be used up to the point where the number of HPN nodes is close to the parallelism factor.

C. IMPACT OF THE RATIO OF MW SERVERS TO HPN NODES

In this set of experiments, we compared the performance of the system with multiple ratio configurations. The ratio between the number of MW nodes and the number of HPN nodes can take multiple values for the same total number of nodes in the system. In this set of experiments, we varied the ratio from one MW node for each HPN node (1/1), to a ratio of one MW node for each 11 HPN nodes (1/11). The total number of nodes in the heterogeneous system was fixed at 13 nodes. The acceleration factor and number of tasks are fixed to 4 and 50, respectively.

Performance results for this set of experiments are illustrated in Fig. 8. Results show that the smaller the ratio, the better the performance. This result is consistent with the expectation that processing on an HPN node is faster than processing on a conventional MW node. The smaller ratios indicate a higher number of HPN nodes. It is worth noting that the higher number of HPN nodes imply additional communication costs between HPN nodes, but with the assumption of sufficient acceleration on the HPN nodes, the performance of the nodes still outweighs the slowness of the communication. When the total number of HPN nodes is larger than the application's parallelism factor, any additional reduction in the ratio will have minimal effect on the performance, since the additional HPN nodes will remain idle. Another factor that affects the ratio parameter is the cost of communication

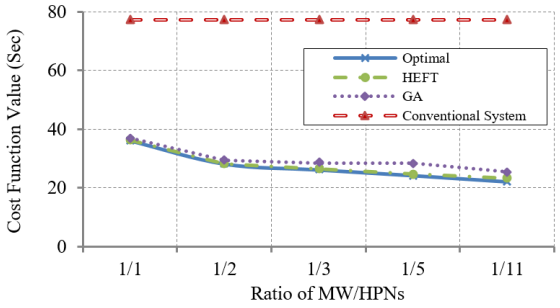

FIGURE 8. Impact of the ratio of MW servers to HPN nodes.

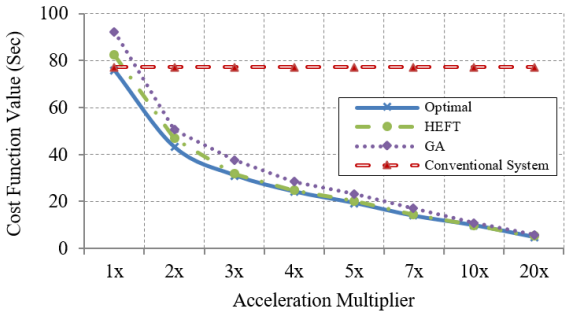

FIGURE 9. Impact of the acceleration multiplier of the HPN nodes.

between the tasks in the DAG. If the average communication cost is high, large ratio values will result in fewer number of HPNs connected to each other through one MW server and most of the data exchanges will need to hop through multiple MW servers. On the other hand, smaller ratio will save some of the communication costs by connecting multiple HPNs to the same MW server.

This set of experiments shows that the best ratio to use is the smallest ratio possible. The smallest ratio corresponds to the largest number of HPN nodes connected to one MW servers and reduces communication costs since HPNs can communicate directly through the common MW server. The smallest ratio may not always be possible to achieve because of the limitation on the maximum number of HPNs that can be connected to one MW server. The limitation is due to the inability to connect many computing nodes to one middleware server without severe degradation in communication performance. For example, geographic locations of HPNs might restrict connecting them to one middleware server.

D. IMPACT OF HPN ACCELERATION

In this set of experiments, we studied the effect of acceleration in the HPN nodes on the performance of the system. We increased the acceleration rate from 1x to 20x while the other parameters (number of tasks, ratio of HPNs to MWs, and number of tasks) are set to the values of 13, 1/5, and 50 respectively. Fig. 9 shows the impact of acceleration on the system performance. The figure also includes as baseline comparison the performance of a conventional system. We assume that a conventional system has the same number of computing nodes as the system with the HPN nodes. However, the compute nodes in a conventional system are homogeneous, where the processing speeds of the compute nodes equal the processing speeds of the middleware servers. Additionally, all the processing nodes in the conventional system are connected to each other with the same bandwidth.

The overall performance improvement of the system follows an exponential growth with the acceleration factor. The nonlinearity in the slope is attributed to the fact that while computation improvements are linearly proportional to acceleration, communication remains proportional to the count and ratio configuration of HPN nodes in the system. Therefore, for DAGs with large communication costs, systems with HPN nodes with small acceleration factors will still give lower performance compared to a system without HPN nodes.

From this set of experiments, we conclude that the best performance is achieved by using the largest acceleration factor possible, which is application dependent. The heterogeneous cloud design configuration can only provide as much parallelism and pipeline streaming execution as the application design can benefit from. Beyond a certain point and according to Amdahl's law, the application is limited by certain task sequencing, and this, in turn, limits the possible acceleration.

E. IMPACT OF THE NUMBER OF APPLICATION TASKS

The characteristics of the application to be executed on the heterogeneous cloud system affect the overall performance. In addition to the parallelism factor discussed earlier, the most important characteristic is the number of tasks. Fig. 10 compares the results of executing different applications with different number of tasks. The plot compares the performance to conventional nodes, and shows that the cost function increases exponentially with the increase in the number of tasks. In this particular example, the application parallelism factor was 15.7. This indicates that the application can benefit, on average, from having 16 compute nodes run in parallel. As the number of tasks increases significantly beyond 16, the application will have more sequential execution, which in turns increases the total execution time. As expected, the plot shows consistent results. When the number of tasks is below 20, the application is able to benefit from parallelism, and a minor rise is observed in the cost of running the application.

F. ALGORITHM PERFORMANCE

Since the algorithm seeks to find an exact solution to the optimization problem, the complexity of the algorithm grows with the increase in problem size. The complexity of the problem is of order $O(n^2)$ where n is the size of the input of the problem. The input of the problem depends on the DAG's number of vertices, DAG's number of edges, DAG's number of parallel nodes, and number of computing nodes.

Fig. 11 shows the effect of increasing the number of nodes and the number of tasks on the complexity of the problem.

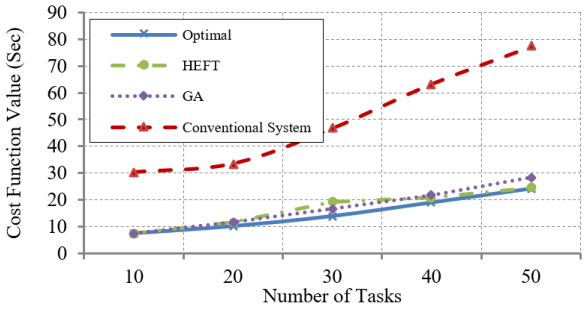

FIGURE 10. Impact of the number of tasks in the DAG.

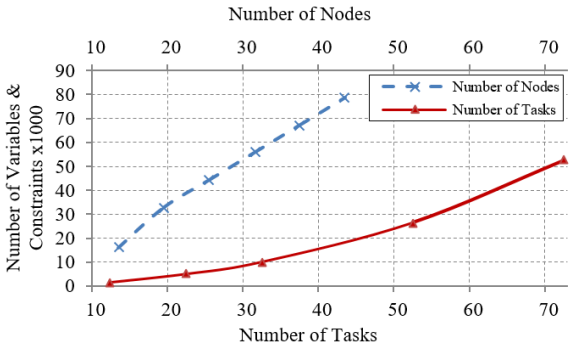

FIGURE 11. Evaluation of the algorithm running performance.

As the number of tasks or nodes increases, the total number of variables and constraints increase impacting algorithm performance. The figure shows that increasing the number of nodes in the system has a more dramatic impact on the complexity of the problem rather more than increasing the number of tasks in the DAG. The algorithm complexity increases exponentially with the increase of either the number of nodes or the number of tasks. As expected, for large problems, finding the optimal solution is not practically feasible. Therefore, heuristics can be used to find the best mapping of tasks to nodes in larger problems. The cost of running the algorithm offline is affordable in the case of repetitive jobs, because small time savings after each iteration add up to a great total latency savings.

The optimal allocation algorithm cannot reach an optimal solution for large problems since the task allocation problem is NP-hard. The optimal search algorithm can be used for mid-size problems to give an initial guess for a suboptimal solution by limiting the time of running the optimal search. In our benchmarking datasets, we assume mid-size problems range from 50 to 150 tasks. Popular choices for heuristics include HEFT and GA. Table I shows the results for running larger datasets (includes DAGs of 100, 200 and 500 tasks). Results show that on large problems the optimal algorithm and after sufficient running time suggest suboptimal solutions close to the results of the heuristics. The table also shows that the heuristics were able to scale to the problem size efficiently. The results of the genetic algorithm

TABLE 1. The percentage degradation from the optimal solutions for large benchmarking dataset.

No.of Nodes	No. of Tasks	HEFT	GA	Suboptimal after 5hr	Suboptimal after 16hr
3	100	215175	213400	221200	216500
	200	321175	320200	-	-
	500	640925	645300	-	-
7	100	59075	63925	59855	59730
	200	88050	109825	-	-
	500	176425	256345	-	-
13	100	32225	40705	48680	31575
	200	47551	58982	-	-
	500	93475	132300	-	-
25	100	17415	24444	-	35342
	200	26326	39400	-	-
	500	52712	87907	-	-

are close to the HEFT algorithm because default crossing and mutation functions were used, and no system specific initialized population were generated.

VII. CONCLUSION

In this paper, the problem of task allocation on the nodes of a heterogeneous and distributed multi-tiered system is first formulated as an optimization problem. An optimal task allocation algorithm is then presented. Since the problem of task allocation is NP-complete, two heuristic algorithms based on the GA and the HEFT are proposed to find feasible solutions for large problems. The analysis of the results provide the following insights about the performance of heterogeneous cloud system:

- Parallelism limit is reached when the number of HPN nodes is close to the parallelism factor.
- Connecting multiple HPNs to the same MW server achieves better performance compared to distributing the HPNs to multiple MW servers.
- In poor acceleration scenarios, the system with HPN nodes perform worse than conventional systems due to the increase in communication costs in the HPN based system.
- The running of the allocation algorithm is directly proportional to the number of tasks in the application and to the total number of computing nodes.
- For large problem sizes, the optimal algorithm fail to reach an optimal solution and could provide suboptimal feasible solution after some run time of the algorithm.
- Heuristics scale on large applications and provide solutions for large problems in which finding an optimal solution is not feasible.

It is worth noting that the proposed optimal scheduling algorithm can be scaled by grouping the computational nodes into clusters and grouping tasks into blocks of tasks. The problem of assigning tasks into computing nodes is transformed into assigning blocks of tasks into computing clusters.

Using this approach, the problem is divided into two tiers: First, allocating blocks of tasks into computing clusters using the scheduling algorithm presented in this paper. Second, for each block of tasks, we allocate the tasks within the block into the computing nodes within the cluster. Hence, the allocation problem is hierarchically addressed by iteratively considering clusters and clusters within clusters.

REFERENCES

[1] *Media Alert: Intel FPGA-Accelerated Storage Technology to be Featured at Flash Memory Summit*, Intel Newsroom, Santa Clara, CA, USA, 2017.

[2] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?" in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2010, pp. 225–236.

[3] S. Huang, S. Xiao, and W. Feng, "On the energy efficiency of graphics processing units for scientific computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, Washington, DC, USA, May 2009, pp. 1–8.

[4] S. Chalamalasetti, M. Margala, W. Vanderbauwhede, M. Wright, and P. Ranganathan, "Evaluating FPGA-acceleration for real-time unstructured search," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2012, pp. 200–209.

[5] L. Gan *et al.*, "Accelerating solvers for global atmospheric equations through mixed-precision data flow engine," in *Proc. IEEE 23rd Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2013, pp. 1–6.

[6] (Nov. 2017). *The Green500 List*. [Online]. Available: https://www.top500.org/green500/lists/2017/06/

[7] (Mar. 2017). *Apache Hadoop*. [Online]. Available: https://hadoop.apache.org

[8] S. Salloum, "Big data analytics on Apache Spark," *Int. J. Data Sci. Anal.*, vol. 1, no. 3, pp. 145–164, Nov. 2016.

[9] *Spark Programming Guide*. Accessed: Jun. 1, 2017. [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html

[10] P. Li, Y. Luo, N. Zhang, and Y. Cao, "HeteroSpark: A heterogeneous CPU/GPU Spark platform for machine learning algorithms," in *Proc. 10th IEEE Int. Conf. Netw. Archit. Storage (NAS)*, Aug. 2015, pp. 347–348.

[11] M. Grossman and V. Sarkar, "SWAT: A programmable, in-memory, distributed, high-performance computing platform," in *Proc. 25th ACM Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2016, pp. 81–92.

[12] Y. Ohno, S. Morishima, and H. Matsutani, "Accelerating Spark RDD operations with local and remote GPU devices," in *Proc. 22nd IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 791–799.

[13] B. Schmidt and A. Hildebrandt, "Next-generation sequencing: Big data meets high performance computing," *Drug Discovery Today*, vol. 22, no. 4, pp. 712–717, 2017.

[14] Y. Yuan, M. F. Salmi, Y. Huai, K. Wang, R. Lee, and X. Zhang, "Spark-GPU: An accelerated in-memory data processing engine on clusters," in *Proc. IEEE Int. Conf. Big Data*, Washington, DC, USA, Dec. 2016, pp. 273–283.

[15] A. Kaitoua *et al.*, "Hadoop extensions for distributed computing on reconfigurable active SSD clusters," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 2, Jun. 2014, Art. no. 22.

[16] M. Huang *et al.*, "Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale," in *Proc. 7th ACM Symp. Cloud Comput.*, Santa Clara, CA, USA, Oct. 2016, pp. 456–469.

[17] H. Artail *et al.*, "Speedy cloud: Cloud computing with support for hardware acceleration services," *IEEE Trans. Cloud Comput.*, to be published.

[18] S. S. Thakur, S. Singh, P. Singh, and A. Goyal, "Optimized task scheduling using differential evolutionary algorithm," in *Advances in Computer and Computational Sciences*. Singapore: Springer, 2017, pp. 509–516.

[19] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci.*, vol. 270, pp. 255–287, Jun. 2014.

[20] M. Shojafar, M. Kardgar, A. A. R. Hosseinabadi, S. Shamshirband, and A. Abraham, "TETS: A genetic-based scheduler in cloud computing to decrease energy and makespan," in *Proc. 15th Int. Conf. Hybrid Intell. Syst. (HIS)*. Cham, Switzerland: Springer, 2016, pp. 103–115.

[21] D. P. Bovet, P. Crescenzi, and D. Bovet, *Introduction to the Theory of Complexity*. London, U.K.: Prentice-Hall, 1994.

[22] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.

[23] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of DAG scheduling heuristics," in *Proc. Integr. Res. Grid Comput. CoreGRID Integr. Workshop*, Hersonissos, Greece, Apr. 2008, pp. 63–74.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[25] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *Proc. 18th Euromicro Conf. Parallel, Distrib. Netw.-Based Process.*, Pisa, Italy, Feb. 2010, pp. 27–34.

[26] A. S. Wu, H. Yu, S. Jin, K. C. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 824–834, Sep. 2004.

[27] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," *J. Parallel Distrib. Comput.*, vol. 70, no. 1, pp. 13–22, Jan. 2010.

[28] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School Comput., Queens Univ., Kingston, ON, Canada, Tech. Rep. 2006-504, Jan. 2006.

[29] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 2, pp. 134–148, Jun. 2014.

[30] P. Visalakshi and S. N. Sivanandam, "Dynamic task scheduling with load balancing using hybrid particle swarm optimization," *Int. J. Open Problems Compt. Math.*, vol. 2, no. 3, pp. 475–488, 2009.

[31] J. Xu, A. Y. S. Lam, and V. O. K. Li, "Chemical reaction optimization for task scheduling in grid computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1624–1631, Oct. 2011.

[32] S. Guo, H.-Z. Huang, Z. Wang, and M. Xie, "Grid service reliability modeling and optimal task scheduling considering fault recovery," *IEEE Trans. Rel.*, vol. 60, no. 1, pp. 263–274, Mar. 2011.

[33] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien, "Scheduling concurrent bag-of-tasks applications on heterogeneous platforms," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 202–217, Feb. 2010.

[34] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Multi-objective resources allocation approaches for workflow applications in cloud environments," in *On the Move to Meaningful Internet Systems: OTM 2012 Workshops* (Lecture Notes in Computer Science), vol. 7567, P. Herrero, H. Panetto, R. Meersman, and T. Dillon, Eds. Berlin, Germany: Springer, 2012, pp. 654–657.

[35] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 3, pp. 338–351, Sep. 2014.

[36] P. K. Yadav, M. P. Singh, and K. Sharma, "An optimal task allocation model for system cost analysis in heterogeneous distributed computing systems: A heuristic approach," *Int. J. Comput. Appl.*, vol. 28, no. 4, pp. 30–37, Aug. 2011.

[37] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci, "Task assignment in heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 66, no. 1, pp. 32–46, Jan. 2006.

[38] Q. Kang, H. He, and H. Song, "Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm," *J. Syst. Softw.*, vol. 84, no. 6, pp. 985–992, Jun. 2011.

[39] G. Attiya and Y. Hamam, "Task allocation for minimizing programs completion time in multicomputer systems," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, in Lecture Notes in Computer Science, vol. 3044, A. Laganá, M. L. Gavrilova, V. Kumar, Y. Mun, C. J. K. Tan, and O. Gervasi, Eds. Berlin, Germany: Springer, 2004, pp. 97–106.

[40] Q.-S. Hua, Z.-G. Chen, and F. C. M. Lau, "A new method for independent task scheduling in nonlinearly DAG clustering," in *Proc. 7th Int. Symp. Parallel Archit., Algorithms Netw.*, May 2004, pp. 187–192.

[41] I. Ahmad, Y.-K. Kwok, and M.-Y. Wu, "Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors," in *Proc. 2nd Int. Symp. Parallel Archit., Algorithms, Netw.*, Jun. 1996, pp. 207–213.

[42] M. Katsev, J. Yu, and S. M. LaValle, "Efficient formation path planning on large graphs," in *Proc. IEEE Int. Conf. Robot. Automat.*, Karlsruhe, Germany, May 2013, pp. 3606–3611.

[43] S. Frey and T. Ertl, "PaTraCo: A framework enabling the transparent and efficient programming of heterogeneous compute networks," in *Proc. 10th Eurograph. Conf. Parallel Graph. Vis.*, Norrköping, Sweden, May 2010, pp. 131–140.
[44] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958.
[45] Z. Wen, J. Cała, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," *IEEE Trans. Services Comput.*, vol. 10, no. 6, pp. 929–941, Nov./Dec. 2017.
[46] Q. Zhu, H. Zeng, W. Zheng, M. Di Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Trans. Emb. Comput. Syst.*, vol. 11, no. 4, Dec. 2012, Art. no. 85.
[47] D. F. da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, "Online task resource consumption prediction for scientific workflows," *Parallel Process. Lett.*, vol. 25, no. 3, pp. 1541003-1–1541003-25, Sep. 2015.
[48] Z. Zhang, L. Cherkasova, and B. T. Loo, "Benchmarking approach for designing a mapreduce performance model," in *Proc. 4th ACM/SPEC Int. Conf. Perform. Eng.*, Prague, Czech Republic, Apr. 2013, pp. 253–258.
[49] S. D. Eppinger, D. E. Whitney, R. P. Smith, and D. A. Gebala, "Organizing the tasks in complex design projects," in *Proc. Comput.-Aided Cooperat. Product Develop. (WCACPD)*. Berlin, Germany: Springer, 1989, pp. 229–252.
[50] H. Z. Jia, A. Y. C. Nee, J. Y. H. Fuh, and Y. F. Zhang, "A modified genetic algorithm for distributed scheduling problems," *J. Intell. Manuf.*, vol. 14, nos. 3–4, pp. 351–362, Jun. 2003.
[51] T. Davidović and T. G. Crainic, "Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems," *Comput. Oper. Res.*, vol. 33, no. 8, pp. 2155–2177, 2006.
[52] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.
[53] C. E. Leiserson, "The Cilk++ concurrency platform," *J. Supercomput.*, vol. 51, no. 3, pp. 244–257, Mar. 2010.

MOHAMMED S. BENSALEH (SM'13) received the Ph.D. degree in electrical engineering from North Carolina Agricultural and Technical State University, Greensboro, NC, USA, in 2005. He is currently an Associate Professor and the Director of the National Center for MEMS Technology, King Abdulaziz City for Science and Technology. His current research interests include cloud computing, low-power design, computer architecture, reconfigurable computing, and wireless sensor networks.

YAMAN SHARAF-DABBAGH received the B.E. degree in electronic and electrical engineering from the University of Aleppo, Aleppo, Syria, in 2011, and the M.E. degree in electrical and computer engineering from the American University of Beirut (AUB), Beirut, Lebanon, in 2014. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, VA, USA. Prior to joining Virginia Tech in 2015, he was a Research Assistant with AUB, where he was involved in the application layer and scheduling for a distributed systems. His current research interests include machine learning, distributed systems, task scheduling, and optimization.

HAZEM HAJJ (SM'08) received the bachelor's degree (Hons.) from the American University of Beirut (AUB) and the Ph.D. degree from the University of Wisconsin-Madison in 1996. Over the years, he has established a strong mix of both industry and academics backgrounds. He joined AUB in 2008, where he is currently an Associate Professor. He was a Visiting Associate Professor with The University of Texas at Austin, Austin, for one year. Before joining AUB, he was a Principal Engineer with Intel Corporation, where he spent 12 years. His research was funded by local and international funding sources, including the funding from Intel Corporation, King Abdulaziz City for Science and Technology, and Qatar National Regional Fund.

He has over 100 research publications in reputable journals and international conferences. His research interests include machine learning and energy-aware computing, with special interests in context-aware sensing, vision systems, and emotion recognition. Over the years, he was a recipient of numerous academic and industry awards.

MAZEN A. R. SAGHIR (SM'12) received the B.E. degree in computer and communication engineering from the American University of Beirut (AUB) in 1989 and the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto in 1993 and 1998, respectively. He is currently a Visiting Associate Professor of electrical and computer engineering at AUB. His research interests include reconfigurable computing, computer architecture, compilers, EDA tools, and embedded systems design.

HAITHAM AKKARY received the B.S. and M.S. degrees in electrical engineering from Louisiana State University in 1981 and 1983, respectively, and the Ph.D. degree from Portland State University in 1998. He is currently a Research Associate Professor with the Electrical and Computer Engineering Department, American University of Beirut. Before becoming a Professor, he was a Principal Research Scientist with Intel Labs. While working at Intel Labs for 20 years, he contributed to the design and development of seven different generations of Intel microprocessors. He has co-authored over 50 technical papers and over 50 U.S. patents. His research interests include microprocessor architecture, architecture support for parallel programming, and computer security.

HASSAN ARTAIL (SM'09) received the B.S. degree (Hons.) and the M.S. degree in electrical and computer engineering from the University of Detroit in 1985 and 1986 respectively, and the Ph.D. degree from Wayne State University in 1999. He is currently a Professor with the Department of Electrical and Computer Engineering, American University of Beirut (AUB), where he is involved in the research in mobile computing, vehicle ad hoc networks, 5G, and Internet of Things. Prior to joining AUB, he was a System Development Supervisor with the Scientific Labs, DaimlerChrysler, MI, USA, where he was involved in the field of software and system development for vehicle testing applications for 11 years. Since joining AUB, he has published over 200 articles in reputable journals and top conferences. He received several awards, including the Research Excellence Award from the Lebanese National Council for Scientific Research in 2012 and the Career Excellence in Scientific Research Award from the Lebanese Association for the Advancement of Science in 2017.

ABDULFATTAH M. OBEID (SM'14) received the Ph.D. degree in electrical and information engineering from the Technical University of Darmstadt, Germany, in 2006. He is currently an Associate Professor with the King Abdulaziz City for Science and Technology and the General Manager of Electronics Unit, TAQNIA.

He played a dominant role in establishing a national IC design center in Saudi Arabia. Under this role, he formed long-term strategic partnerships with a number of international private and public organizations for the implementation of various projects of national importance. These projects were mainly related to the design and development of digital, mixed-signal ASICs, and inertial MEMS sensors for different applications. He was a Key Member (Coordinator of the electronics, communications, and photonics strategic technology) of the Implementation Committee that drafted the National Science, Technology and Innovation Plan for Saudi Arabia.

He has co-authored several technical papers in refereed international conferences and journals. His research interests include hardware/software co-design, reconfigurable computing, computer architecture, and wireless sensor networks. He is a member of ACM. He has served on the technical committee of several IEEE conferences and workshops.

SYED MANZOOR QASIM (SM'12) received the B.Tech. and M.Tech. degrees (Hons.) in electronics engineering from the Z. H. College of Engineering and Technology (ZHCET), Aligarh Muslim University, India, in 2000 and 2002, respectively. He is currently working as a Senior Research Consultant at King Abdulaziz City for Science and Technology (KACST). Prior to this, he was a Research Engineer with King Saud University, a Consultant with the Technology Development Center, KACST, and the Manager of Research and Development with the National Center for Electronics and Photonics Technology, CITRI, KACST. He has published over 100 papers in peer-reviewed journals and conferences. He has three U.S. patents, with several other pending patents. His current research interests include system-on-chip design, digital IC design, computer architecture, reconfigurable computing, FPGA design, machine learning, cloud computing, and Internet of Things. He is a Life Member of the Institution of Electronics and Telecommunication Engineers, India, and the International Association of Engineers, Hong Kong.