AMERICAN UNIVERSITY OF BEIRUT

Task Scheduling in a Reconfigurable Active SSD Distributed System

by
YAMAN SHARAF DABBAGH

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
December, 2014

AMERICAN UNIVERSITY OF BEIRUT

TASK SCHEDULING IN A RECONFIGURABLE ACTIVE SSD DISTRIBUTED SYSTEM

by

YAMAN SHARAF DABBAGH

Approved by:

Dr. Hazem Hajj, Associate Professor — Advisor
Electrical and Computer Engineering

Dr. Hassan Artail, Professor — Member of Committee
Electrical and Computer Engineering

Dr. Haitham Akkary, Associate Professor — Member of Committee
Electrical and Computer Engineering

(on behalf of Dr. Akkary)

Dr. Mazen A. R. Saghir, Associate Professor — Member of Committee
Electrical and Computer Engineering
Texas A&M University at Qatar, Doha, Qatar

(on behalf of Dr. Saghir)

Date of thesis defense: December 11, 2014

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

Student Name: Sharaf Dabbagh Yaman

Last First Middle

● Master's Thesis ○ Master's Project ○ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☒ I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project,** to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

Yaman Sharaf Dabbagh Feb 19th, 2015

Signature Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

Acknowledgments

Family, and Friends

An Abstract of the Thesis of

Yaman Sharaf Dabbagh for Master of Engineering
Major: Electrical and Computer Engineering

Title: Task Scheduling in a Reconfigurable Active SSD Distributed System

In massive computing applications that process large amounts of data, I/O operations consume a significant portion of the overall execution time. Such applications belong to the class of Big Data Analytics (BDA). High-end computational capabilities and active storage solutions focus on pushing as much of the computation as possible closer to where the data resides. This decreases the time taken to transfer the data from the storage to the computing nodes, thus reducing the overall time taken by I/O operations. However, other components of the computations, such as data aggregation and subsequent processing of aggregated data, are frequently serial or exhibit low levels of parallelism, thus requiring these components to execute on high-performance general-purpose processing nodes. Although the resulting distributed computing systems required for executing BDA applications feature massive parallel processing of huge amount of data on specialized computing nodes, they also feature significant complexity in scheduling multiple application components with various computational requirements for optimal execution on heterogeneous computing nodes. The heterogeneity in the system is due to both having different types of computing nodes and using multiple

types of connecting networks between the computing nodes. This paper considers cloud environments integrated with high-end active storage systems, and aims to develop tools and methods for optimal distribution of computational tasks of BDA applications in these heterogeneous computing systems. We evaluate mapping of big data applications represented in Directed Acyclic Graphs (DAG) into the heterogeneous computational nodes. This mapping is done through an optimization algorithm that minimizes the overall execution time and data communication delay. We compare the algorithm with the Genetic Algorithm (GA) and Heterogeneous Earliest Finish Time (HEFT) algorithms. We compare the performance between a standard system and the system with high-performance nodes by applying the same task scheduling algorithm on both systems. Finally, we propose methods for task allocation on large problem sizes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Scientific and analytical applications in high performance computing systems contain massive amounts of data with intensive I/O operations. These I/O operations are considered one of the top challenges in data-intensive applications (the number one challenge in extreme-scale visual analytics [**1**]). One of these terascale applications was implemented over the 5^{th} fastest supercomputer of 2008 (IBM Blue Gene/P) and at this scale the major challenges were I/O related [**2**]. Active storage systems have been proposed to overcome this problem [**3**] by moving computation closer to the storage nodes and reduce the cost of I/O operations. The active storage concept has been a major focus and will gain even more attention in the coming years based on the International Exascale Software Project community roadmap [**4**]. The Reconfigurable Active Solid State Drive system (the RASSD system) [**5**] is one of the recent proposed active storage systems. The RASSD system consists of three layers: application layer, middleware servers layer, and High Performance Nodes (HPNs) layer. Figure 1.1 shows a sample RASSD system with a client, 3 middleware servers, and 5 HPNs. The application layer (also called the client) is where the user communicates with the system to run data processing

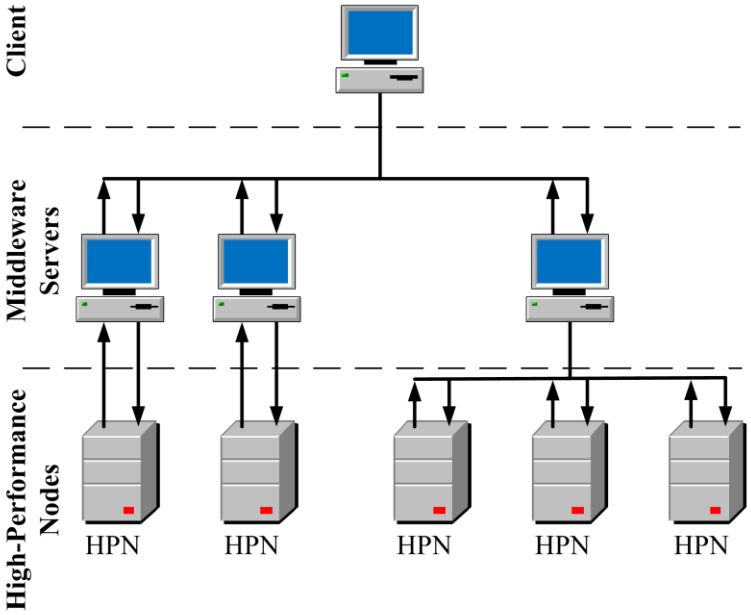


Figure 1.1: Sample RASSD system

queries, and receive and view results. The middleware servers layer receives the requests from the application layer and acts as a management middle stage between the application and HPN layers, it handles scheduling, task allocation, communication, and aggregation operations. The HPN layer which forms the core component of the system consists of FPGA processors connected to SSD storage drives, FPGA processors along with the SSD drives forms the active storage component in the system. The main question that rises for all the active storage systems is how to program these systems? [**3**]. In other words, given an application of n tasks and a system of m computational nodes, what is the assignment of tasks n to nodes m that achieves the best performance? In this paper, we present an allocation algorithm that provides the optimal assignment of tasks to computational nodes. For our knowledge no one has presented a programming model for the RASSD system or a heterogeneous multi-tiered system with high performance nodes. In similar active storage systems the developer assigns the applications'

tasks to each computing node in the active storage system [**6**], like in the IBM netezza project [**7**] that uses also FPGA processors to perform computation directly on the data. The challenges in automating the assignment of tasks to nodes come from the complexity of the architecture of the RASSD system. The complexity in the RASSD system comes from two reasons: 1) the heterogeneous architecture, where nodes in the system has different computing speeds. 2) the different types of connections between computing nodes in the system. For example the connection speed between HPNs is different than the connection speed between MWs. Task scheduling algorithm is an NP-hard problem. Therefore, optimal task assignment is only possible for relatively small problems, whereas only heuristic solutions can be derived for relatively large problems.

This paper makes the following key contributions:

- An optimal allocation algorithm that maps tasks of an application to computational nodes in the RASSD system.
- Apply the proposed algorithm on both a standard system and the RASSD system, and analyze the strengths and weaknesses of the RASSD system compared to other systems.

Compare suboptimal solutions of the algorithm when applied on large problems with the Genetic Algorithm (GA) and Heterogeneous Earliest Finish Time (HEFT) heuristics.

The reminder of the thesis is organized as follows. We review the related work in Chapter 2. Chapter 3 describes in details the RASSD system topology and scheduling model. In Chapter 4 we present our optimal allocation algorithm. The GA algorithm used in the comparison is detailed in Chapter 5. The results and analysis is in Chapter 6. At last we conclude in Chapter 7.

Chapter 2

Literature Review and Related Works

In this chapter, we include a brief summary of the architecture of the RASSD system. Then, a background on the mathematical theories used for scheduling in this thesis. The related work is divided into two parts. The first part introduces the related works based on the scheduling algorithms available in the literature. The second part introduces the recent distributed computing platforms similar to the RASSD system and how scheduling is implemented on these systems.

2.1 The RASSD Environment

RASSD environment [**5**] consists of three layers shown in Figure 1.1:

Application layer: which is the highest level layer and the one the client communicates with to run data processing queries. These requests are sent to the lower level called the middleware layer.

Middleware layer: consists of general purpose machines that receive the request from the application layer and determine the location of the data that the query re-quested and also determine the operations that need to be applied over the data.

Hardware layer: which forms the core component of the system that consists of FPGA processors connected to SSD storage drives. The operating system installed on the FPGA receives from the middleware layer the information that tells the FPGA what type of operations are needed to be performed and on what data these operation will be performed.

2.2 Mathematical Programming

The problem of scheduling, which is allocating a set of tasks into a set of compute nodes, has many techniques to solve it. Mathematical programming is one of the tools to solve decision making problems like the scheduling problem. Mathematical programming is concerned in achieving an objective while being restricted with certain constraints on the resources available.

The objective function in mathematical programming takes the form of:

$$\text{Minimize (or Maximize)}\, f(x)$$

Where x is the set of decision variables for optimization problem. Therefore, the objective is to find the values of the decision variables x which minimizes or maximizes the objective function which is usually a linear function of the decision variables. The decision variables could be bounded by a set of constraints. The mathematical form of the constraints is:

$$f_1(x) \leq b_1$$

$$f_2(x) \leq b_2$$

$$f_m(x) \leq b_m$$

Where $f_1, f_2, ..., f_m$ is a set of functions of the decision variables x, and $b_1, b_2, ..., b_m$ is a set of constants. The problem is considered linear programming when the objective function and constraints are all linear. Solving integer problems is easier than solving non-linear problems because non-linear problems have a non-convex shape for the functions and hence searching for the global solution is impossible. In the problem of scheduling the assignment decision variables are binaries. Therefore, a modified version of the linear programming is used called Mixed Integer Linear Programming (MILP). The first step to solve the MILP problem is to solve the linear relaxation of the problem. If the relaxed LP problem has a feasible solution this means the MILP problem has a solution bounded by the LP solution. One of the algorithm to search for a solution for the MILP problem is the branch and bound algorithm. The branch and bound algorithm uses the LP techniques to find upper and lower bounds for the problem and recursively tries to make the gap between the two bounds smaller. Each non-integer decision variable is analyzed and then two sub-problems are created for two values for the non-integer decision variables, these two sub-problems are called branches. Another variable is analyzed for each branch recursively.

2.3 Related Works

Scheduling in distributed systems can be classified into different categories as described in [**8**]. 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

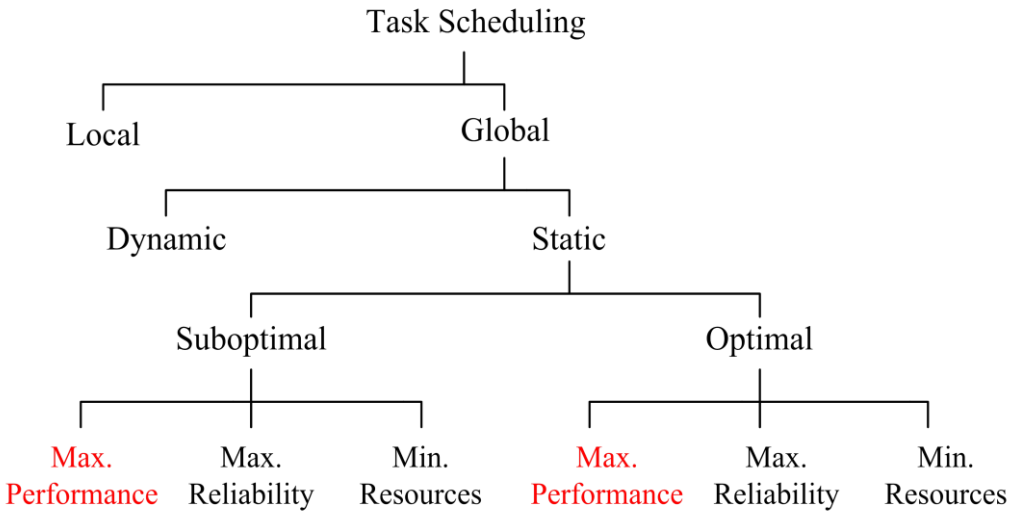


Figure 2.1: Block diagram showing the setup for the algorithm

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. While 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. Figure 2.1 shows the taxonomy presented before. Our work in this paper is in the global-static scheduling. We propose an optimal algorithm to achieve the best performance along with implementation of two heuristics for performance maximization.

Research work in [**9**] [**10**] [**11**] [**12**] [**13**] [**14**] focus on optimal scheduling of tasks to enhance the execution. In [**9**] the communication links were considered

identical and the authors used the meta-heuristic Hybrid Particle Swarm Optimization (HPSO) to solve the formulation. In [**10**] the authors 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 [**11**] reliability and performance were maximized, and Ant Colony Optimization (ACO) was used in this work. In [**12**] the author 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 a maximum of one group of tasks. In [**13**] the objective function were formulated to minimize both the execution time of the application along with minimizing the workflow of the application. In [**14**] the authors proposed an optimization formulation for a heterogeneous system of computing nodes used in automobiles. Their approach to build the formulation is used in our work to build the formulation for the specific RASSD system.

Heuristic approaches were presented in [**15**] [**16**] [**17**] [**18**] [**19**] [**20**] [**21**] [**22**] [**23**]. In [**15**] the authors proposed a greedy heuristic scheduling algorithm. The heuristic allocates the tasks with the heaviest communication links on the fastest link available and so on. In [**16**] the authors implemented 6 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 between the rest of the algorithms. In [**17**] the authors proposed an iterative greedy algorithm to allocate tasks to enhance performance. In [**18**] the authors implemented the Simulated Annealing (SA) heuristic algorithm to find out the best performance of the system taking in consideration the constraints of the system. In [**19**] [**20**] they clustered the tasks into different groups where each group is implemented later on a computing node

of the parallel system. In [**21**] [**22**] [**23**] graph mining techniques were used to come up with the best task allocation.

Scheduling algorithms can be categorized also based on the types of distributed systems. We will organize the available distributed platforms into three categories based on its similarity with the RASSD environment:

2.3.1 Standard Distributed Platforms

In this category, we organize all the distributed computing platforms consisting of CPUs and storage resources. These platforms have many types and can be classified based on different criteria [**24**] [**25**] [**26**] (hardware used, type of interconnection network between nodes, and type of service provided for the user). Examples of these platforms are the Amazon EC2 [**27**], Microsoft Windows Azure platform [**28**], and Google App Engine [**29**]. These platforms share with the RASSD platform the same general structure (compute and storage components and interconnection network). Designing programs for these platforms differs from one platform to another. Some follow the general Message Passing Interface (MPI) model or a platform-dependent model like MapReduce programming model [**30**] for Google App Engine. Distributing tasks among nodes in these systems is either left for the user to decide or integrated in the platform. For example MapReduce running platforms hides the process of task allocation from the user and implements load balancing between computing nodes in the background.

2.3.2 Reconfigurable Distributed Platforms

Reconfigurable Distributed Platforms use FPGAs beside CPUs to enhance the performance [**31**] [**32**]. In these systems FPGAs or boards containing many

FPGAs are connected to CPUs or microprocessors to form a distributed platform. Examples of these platforms are the SRC-6E by SRC Computers LLC [**33**]and Cray XT4-5 by CRAY the supercomputer company. Writing programs for these systems depends also on the middleware running on the CPUs and handling the communication signals between them. And for programming the FPGAs many programming models exist [**34**] like VHDL and Synchronous Dataflow using Simulink. In most cases task allocation is left for the developer to decide on.

2.3.3 Active Storage Distributed Platforms

In this category we include the platforms that uses active storage nodes (apply computation directly on the data). Platforms in this category differ based on the active storage devices used. Some platforms dedicate the computing nodes which are directly connected with the storage as active storage nodes (or so called staging nodes) [**35**] [**36**] [**37**]. Other platforms use specialized hardware for the active storage nodes like GPUs [**35**] or FPGAs [**7**]. Programming models for these platforms tend to partition the application into two parts, one to run on the compute nodes and the other to run on the active storage nodes. Programming the compute-node part is similar to the standard platforms previously described, where programming the active storage nodes depends on the type of the hardware used. In the case where we have FPGAs as active storage nodes, programming these active storage nodes can be done in different ways [**34**] just like programming platforms in the previous category (reconfigurable distributed platforms).

From the related work, we can see that there has been a little work on task scheduling on system with active storage nodes. To the best of our knowledge, no

scheduling work has been done on a multi-tier system with high performance nodes.

Chapter 3

Scheduling Model

The objective of this work is to determine an optimal distribution of tasks over a distributed cloud computing environment with specialized compute node capable of high acceleration. Towards deriving the optimization formulation, the problem can be described as shown in Figure 3.1. 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 RASSD System Topology, Data location, and Directed Acyclic Graphs (DAG).

3.1 RASSD System Topology

The RASSD system topology includes information about the number of physical nodes, and the network topology which is the arrangement of physical nodes and how 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). Figure 3.2 shows the RASSD system topology

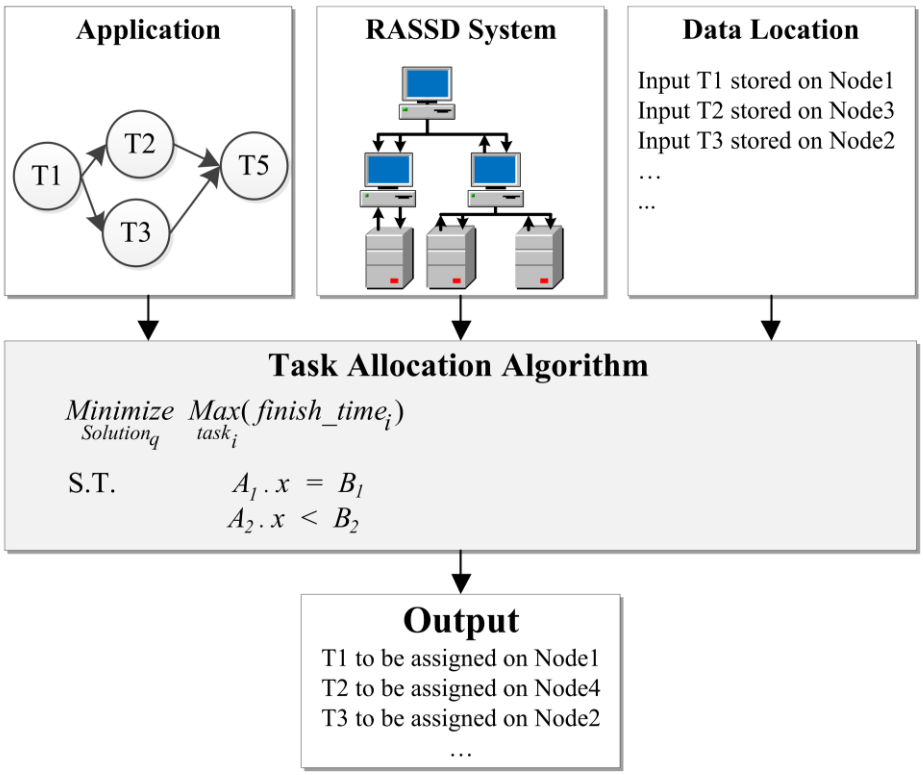


Figure 3.1: Block diagram showing the setup for the algorithm

for a system with 5 HPN nodes and 3 Middleware servers. Figure 3.2 also shows the possible 7 links in the system:

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.

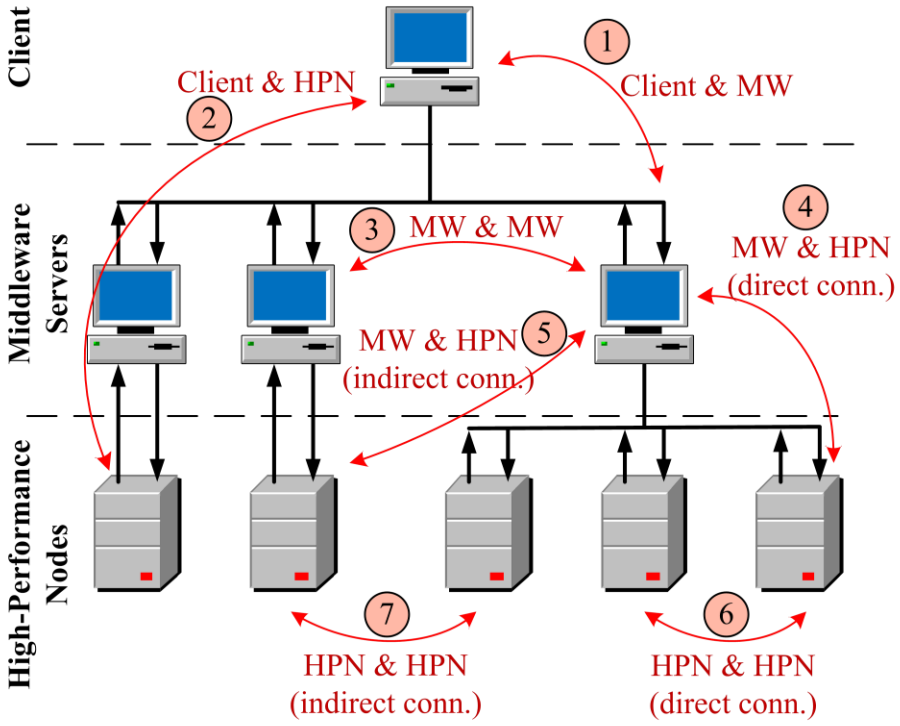


Figure 3.2: The possible links in the RASSD system

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).

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).

3.2 Data location

Data location includes information about the storage locations for the input data of each task.

3.3 Directed Acyclic Graphs (DAG)

DAGs are used to represent applications. Each vertex in the DAG represents a compute node and has a weight representing the execution time of the task on that node. Each edge has a weight representing the communication time between the two compute nodes connected by the edge. A DAG typically has a single entry node and a single exit node. An entry node is a node with no parent. An exit node is a node with no child. We make the following assumptions to 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 normal processing nodes and high performance RASSD compute nodes. The weights associated with RASSD

nodes are lower to reflect accelerated processing and faster execution times,.

The 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 RASSD node, an additional vertex is added to reflect the data read or write task on that node. The weights of the vertices for the read and write tasks are adjusted to reflect localization of the data.

Chapter 4

Scheduling Algorithm

The objective of the optimization algorithm is to map the application tasks to the RASSD 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:

$$\text{Min}\, f(x)$$

Subject to:

$$A\,x \leq b$$

$$A_{eq}\,x = b_{eq}$$

$$x \geq 0$$

Where $x = x_1,...,x_n$ 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 num-

ber 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 RASSD system and the DAG into the optimization problem.

4.1 Variables and Parameters

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

4.1.1 Input Parameters

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 $i, j \in T$

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

$P_{i,j}$ 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 RASSD 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 allocaed to nodes $a,b \in N$ respectively. Note that this depends on $C_{i,j}$ and the physical communication between the two nodes a,b .

RASSD System representation of physical nodes:

N Set of all the physical Nodes in the RASSD 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 RASSD system.

MW Set of (labels for) all **M**iddle**W**are servers in the RASSD 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 $m \in MW$

4.1.2 Supporting Decision Variables

$A_{i,j,a,b}$ 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.

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

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

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

4.1.3 Output Decision Variables

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

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

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.

4.2 Objective Function

The objective function characterizes the optimal solution. The optimal solution ensures executing the given application represented as a DAG on the given RASSD 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.

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

Subject to:

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

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

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

The size of the vector n is dependent on the number of variables for allocations of tasks to nodes:

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

Where: number of possible types of physical links = 7 for the RASSD system

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

4.3 Constraints

The first constraint is the smoothing constraint already mentioned before:

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

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 in serial. 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:

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

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.

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

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

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

$$\begin{aligned} S_j - e_{i,a} \times A_{i,a} - S_i < M \times \ 1 - O_{i,j} \\ \forall \ \ i,j \ \ \in T \ | P_{i,j} = 0 \end{aligned} \tag{8}$$

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 $O_{i,j} \neq O_{j,i}$. Therefore, we add the following constraint:

$$\begin{aligned} A_{i,a} + A_{j,a} + O_{i,j} + O_{j,i} \leq 3 \\ \forall \ \ i,j \ \ \in T \ , \forall a \in N \mid P_{i,j} = 0 \end{aligned} \tag{9}$$

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

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

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:

$$\begin{aligned} \sum_{a,b \ \in N} A_{i,j,a,b} \leq 1 \\ \forall \ \ i,j \ \ \in T, \forall \ \ a,b \ \ \in N \mid L_{i,j} = 1 \end{aligned} \tag{11}$$

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 $A_{i,j,a,a}$):

$$\begin{aligned} 2 - A_{i,a} - A_{j,a} \geq \sum_{a,b \ \in N} A_{i,j,a,b} \\ \forall \ \ i,j \ \ \in T \ , \forall \ \ a,b \ \ \in N \mid L_{i,j} = 1 \end{aligned} \tag{12}$$

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 equal to 1 indicating they are physically connected. Furthermore, for the case of where both of 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:

$$\sum_{a\in HPN_m} A_{i,a} + \sum_{b\in HPN_m} A_{j,b} - 1 \le A_{i,j,a,b} \quad \forall\ i,j\ \in T,\forall m \in MW, a \in HPN_m, b \in HPN_m \left| L_{i,j} \right\rangle 0 \tag{13}$$

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 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 $m \neq n$:

$$\sum_{a\in HPN_m} A_{i,a} + \sum_{n\in MW|n\neq m} \sum_{b\in HPN_n} A_{j,b} - 1 \le A_{i,j,a,b} \tag{14}$$

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

If the one of the consecutive tasks i,j is allocated to an **HPN** node and the other task is allocated to a **middleware**, then the allocation variable $A_{i,j,a,b}$ must 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:

$$\sum_{a\in HPN_m} A_{i,a} + A_{j,m} - 1 \le A_{i,j,a,m} \tag{16}$$

$$A_{i,m} + \sum_{a\in HPN_m} A_{j,a} - 1 \le A_{i,j,m,a} \quad \forall\ i,j\ \in T,\forall m \in MW, a \in HPN_m \left| L_{i,j} \right\rangle 0 \tag{17}$$

If the one of the consecutive tasks i, j is allocated to an **HPN** node and the other task is allocated to a **middleware**, then the allocation variable $A_{i,j,a,b}$ must 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 , $m \neq b$:

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

$$\begin{gathered} \sum_{n \in MW | n \neq m} A_{i,n} + \sum_{a \in HPN_m} A_{j,a} - 1 \leq A_{i,j,n,a} \\ \forall \ \ i, j \ \ \in T, \forall \ \ m, n \ \ \in MW, a \in HPN_m \left| n \neq m, L_{i,j} \right\rangle 0 \end{gathered} \tag{19}$$

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 equal to 1 indicating they are physically connected:

$$\begin{gathered} \sum_{m \in MW} A_{i,m} + \sum_{n \in MW} A_{j,n} - 1 \leq A_{i,j,m,n} \\ \forall \ \ i, j \ \ \in T, \forall \ \ m, n \ \ \in MW \left| L_{i,j} \right\rangle 0 \end{gathered} \tag{20}$$

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

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

$$\begin{gathered} A_{i,a} + \sum_{m \in MW} A_{j,m} - 1 \leq A_{i,j,a,m} \\ \forall \ \ i, j \ \ \in T, m \in MW \left| L_{i,j} \right\rangle 0, a = CL \end{gathered} \tag{22}$$

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 equal to 1 indicating they are physically connected:

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

$$\begin{aligned} A_{i,a} + \sum_{m \in MW} \sum_{b \in HPN_m} A_{j,b} - 1 \leq A_{i,j,a,b} \\ \forall \ \ i,j \ \ \in T, m \in MW, b \in HPN_m \left| L_{i,j} \right\rangle 0, a = CL \end{aligned} \tag{24}$$

Chapter 5

Comparison of Heuristics

5.1 Genetic Algorithm

Since the scheduling task is an NP-Hard problem. A heuristic algorithm has to be used for large problems. Genetic Algorithms (GAs) have been widely used and proven to obtain high quality solutions. The Genetic Algorithm 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 unlike the optimization algorithm.

5.1.1 Initial Population and Representation of Solutions

The most commonly used representation of solutions is an array of tasks for each machine [**38**]. For simplicity we will represent each machine with a number from

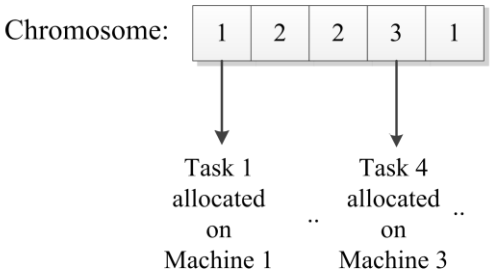


Figure 5.1: Sample chromosome for a DAG of 5 tasks and a system of 3 physical nodes

1,…, N where N is the total number of machines in the RASSD 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 Figure 5.1.

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: 1) Suboptimal Algorithm: using the same optimal algorithm described in section 4 even on large problems feasible solutions are given if optimal solution was not found. 2) Manual allocation on the HPN: since the RASSD system includes HPN nodes with accelerated processing adding initial solutions where all tasks are allocated on the HPN nodes. 3) Solutions to other heuristics: HEFT allocation solution is also added to the initial population.

5.1.2 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

5.2 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.

Chapter 6

Experiments and Results

The objective of the experiments is to evaluate the performance of the optimization algorithm. The proposed algorithm to allocate tasks to compute nodes is compared to the performance of the Genetic Algorithm (GA), Heterogeneous Earliest Finish Time (HEFT), and the same algorithm but on a conventional system. The following factors are considered:

1. Hardware system configurations:
    a. Number of processing nodes: takes values from 3 to 37 processing nodes.
    b. Ratio of HPN nodes to MW servers: takes values 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.
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.

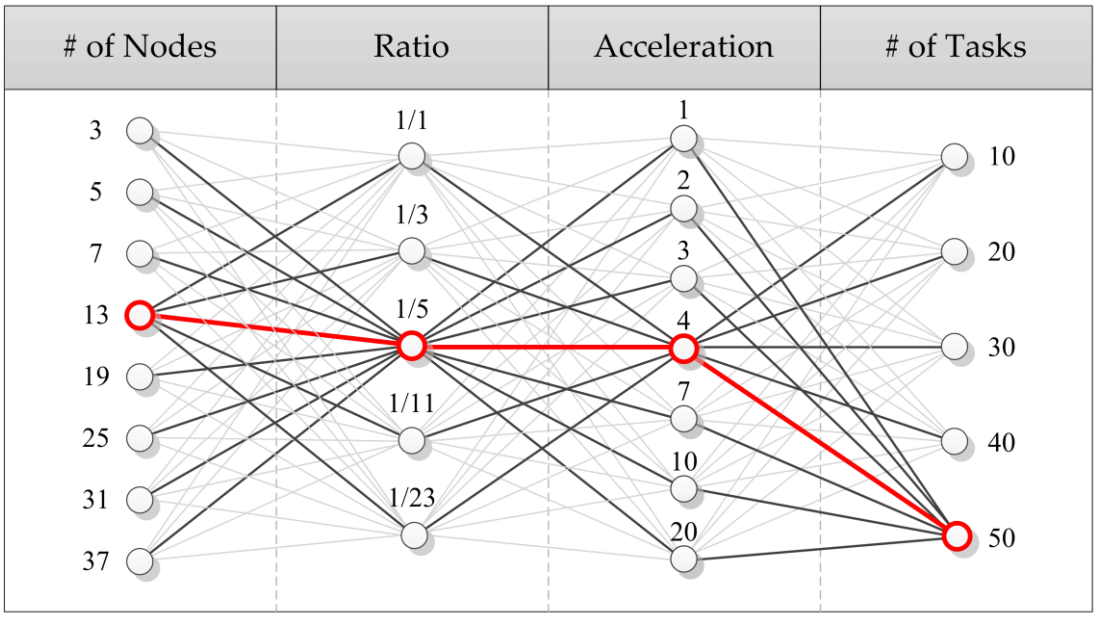


Figure 6.1: Set of conducted experiments

Figure 6.1 shows the set of the conducted experiments. The first three columns represent the different options for hardware configuration. The forth 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.

6.1 Experimental Setup

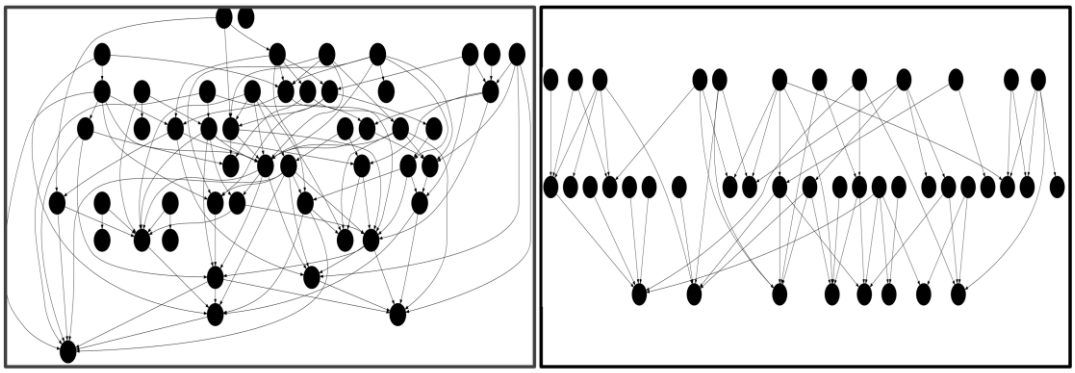
The experiments were performed on a set of benchmarking DAGs generated by Davidovic and Crainic [**39**]. The DAGs provided by Davidovic feature both task execution costs and communication costs, whereas other benchmarking DAGs like the Standard Task Graph set (STG) [**40**] do not include communication costs in their DAGs tasks. Each test case was repeated 10 times and the average cost function value was calculated. IBM ILOG CPLEX [**41**] was used as the Mixed Integer Linear Programing (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 [**42**]. 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 is 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 the 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 (∞)* is the span that consists of the time needed to run the most expensive path in the application that extends from the beginning to the end of the application; it is also called the critical path length.



(a) *Parallelism=4.9793* (b) *Parallelism=15.4*

Figure 6.2: Parallelism in DAGs. (a) is a DAG with 50 tasks and parallelism factor of 4.979. (b) is a DAG with 50 tasks and parallelism factor of 15.4

Figure 6.2 provides a visual illustration of the parallelism in a DAG. Both DAGs in Figure 6.2a and Figure 6.2b have the same number of tasks, but the DAG in Figure 6.2a has more dependencies than the DAG in Figure 6.2b. As a result, the DAG in 5a has more sequential tasks. These differences are reflected in the level of parallelism. The DAG in 5a has a parallelism factor of 4.9793 where the DAG in Figure 6.2b has a 15.4 parallelism factor. The DAG in Figure 6.2b has a maximum of 3 sequenced tasks shown in the figure as 3 levels of tasks.

6.2 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 Figure 6.3, which shows that the RASSD system is on average 2 to 3 times better than a conventional system without the RASSD nodes. The improvement in performance takes an exponential instead of linear

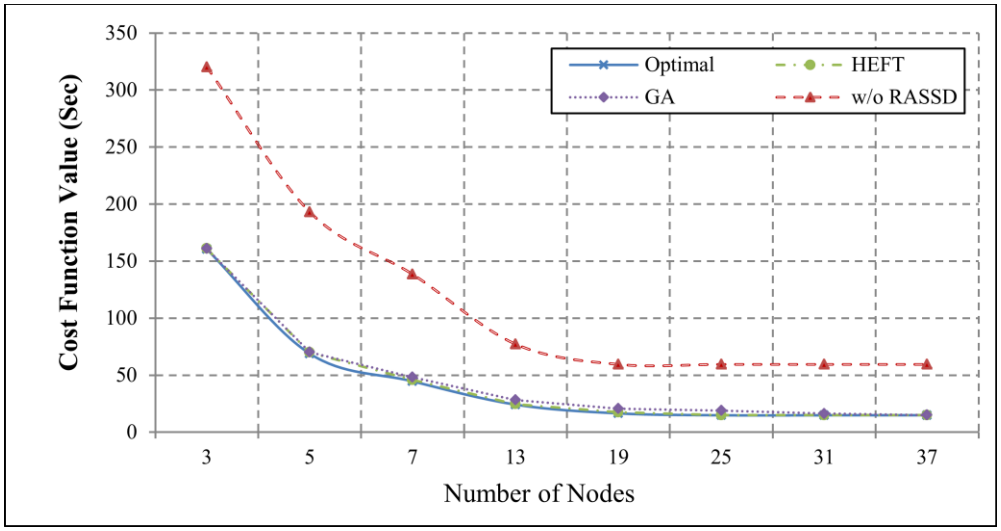


Figure 6.3: Impact of increasing the number of computing nodes

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. 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 RASSD 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.

6.3 Impact of the Ratio of MW servers to HPN nodes

In this set of experiments, we compare the performance of the system with multiple ratio configurations. The ratio between the number of Middleware 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 Middleware node for each HPN node (1/1), to a ratio of one Middleware node for each 11 HPN nodes (1/11). The total number of nodes in the RASSD system was fixed at 13 nodes. Acceleration and number of tasks are fixed to 4 and 50 respectively.

Performance results for this set of experiments are depicted in Figure 6.4. 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 middleware node. The smaller ratios indicate a high-

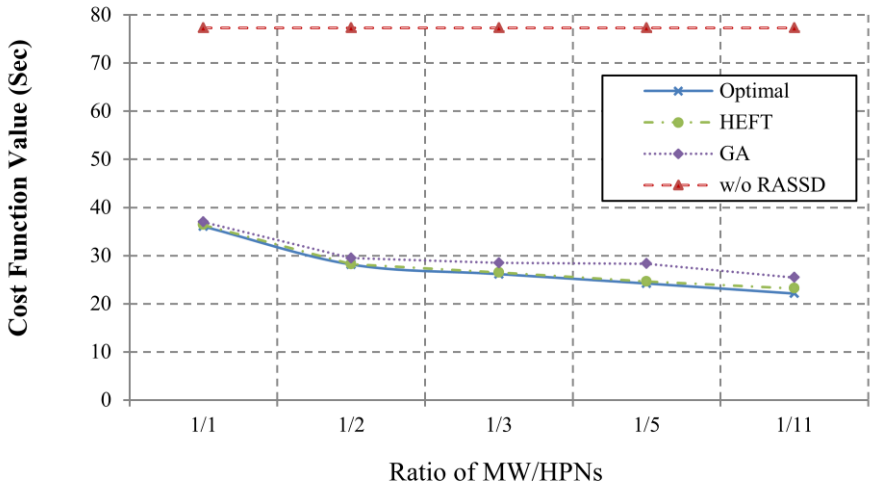


Figure 6.4: Impact of the Ratio of MW servers to HPN nodes

er number of HPN nodes. It is worth noting that the higher number of HPN nodes implies additional communication costs between HPN nodes, but with the assumption of sufficient acceleration on the HPN nodes, the performance of the nodes still outweigh 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 which affects the ratio parameter is the cost of communication 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 signals will need to hop through multiple MW servers. Whereas 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; it provides 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, or the limitation due to the locations of the HPNs and the inability to physically connect them to one MW server.

6.4 Impact of HPN Acceleration

In this set of experiments, we study the effect of acceleration in the HPN nodes on the performance of the system. We increased the acceleration rate from 1x to 20x whereas 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. Figure 6.5 shows the impact of acceleration on system performance. The figure also includes as baseline comparison the performance of a conventional system.

The overall performance improvement of the system follows an exponential rate 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 RASSD design configuration can only provide as much parallelism and pipeline streaming execution as the application design can benefit from. Be-

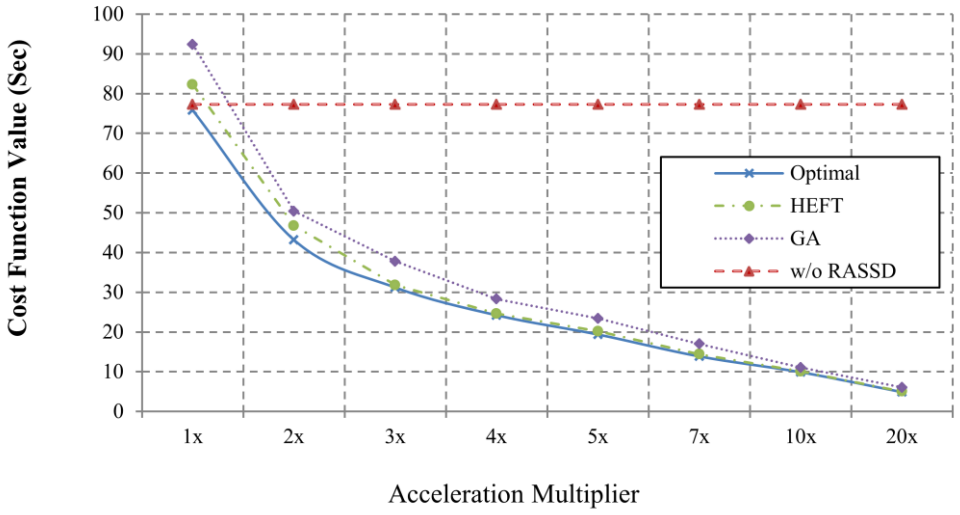


Figure 6.5: Impact of the Acceleration Multiplier of the HPN nodes

yond a certain point, the application is limited by certain task sequencing, and this, in turn, limits the possible acceleration.

6.5 Impact of the Number of Application Tasks

The characteristics of the application to be executed on the RASSD system affect the overall performance. In addition to the parallelism factor discussed earlier, the most important characteristic is the number of tasks.

Figure 6.6 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

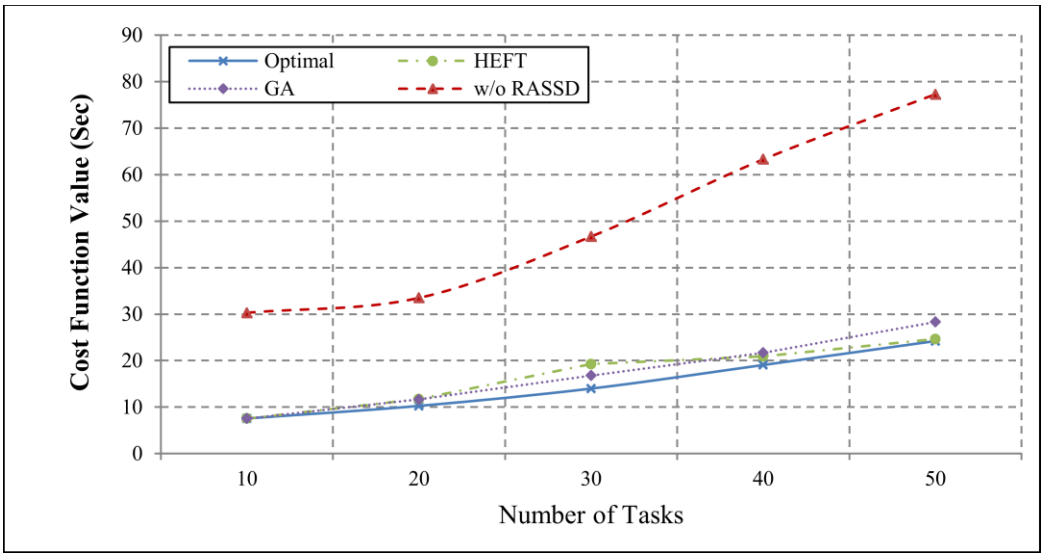


Figure 6.6: Impact of the Number of tasks in the DAG

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 increase is observed in the cost of running the application.

6.6 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.

Figure 6.7 shows the effect of increasing the number of nodes and the effect of increasing the number of tasks on the complexity of the problem. As the number of tasks or nodes increase the total number of variables and constraints increase impacting algorithm performance. The figure shows that increasing the number of

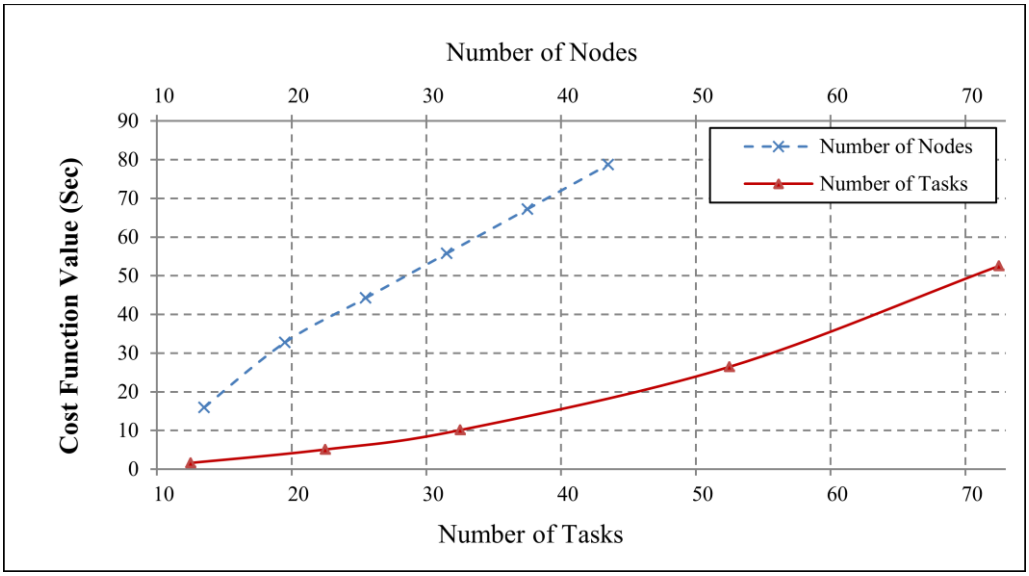


Figure 6.7: Evaluation of the Algorithm Running Performance

Table 6.1: The percentage degradation from the optimal solutions for large benchmarking datasets

#of Nodes	#of Tasks	HEFT	GA	Suboptimal after 6hr	Suboptimal after 17hr
3	100	7.59	6.7	0.5	
	200	3.6	3.29	-	-
	500	1.73	2.42	-	-
5	100	1.09	5.49	0.54	
	200	14.75	39.86	-	-
	500	5.96	48.88	-	-
7	100	1.85	10.21	0.86	
	200	3.59	29.20	-	-
	500	10.27	60.22	-	-
13	100	3.95	31.31	3.23	
	200	5.62	33.5	-	-
	500	15.92	41.26	-	-

nodes in the system increases the complexity of the problem rather more than increasing the number of tasks in the DAG. The algorithm performance increases exponentially with either increase in the number of nodes or 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 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. Popular choices for Heuristics include Heterogeneous Earliest Finish Time (HEFT), and Genetic Algorithm (GA). Table 6.1 shows the results for running larger datasets (100, 200, and 500 tasks DAGs).

Chapter 7

Conclusion

In this paper, we have presented an optimal algorithm to allocate tasks of an application to nodes of a distributed multi-tiered system with high performance nodes. Also, we have implemented two heuristic algorithms GA and HEFT and compared their performances with the optimal algorithm for small application sizes. As for larger programs the optimal algorithm does not scale whereas heuristics scale. The analysis of the results gave us the following insights about the RASSD system performance:

- 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.
- Performance of the algorithm was compared with two heuristics the genetic and heterogeneous earliest finish time algorithms.
- Implemented heuristics for large applications to provide solutions for large problems in which finding an optimal solution is not feasible.

Bibliography

[1] P. C. Wong, H. W. Shen, C. R. Johnson, C. Chen, and R. B. Ross, "The top 10 challenges in extreme-scale visual analytics," *Computer Graphics and Applications, IEEE*, vol. 32, pp. 63-67, 2012.

[2] R. T. Fisher et al., "Terascale turbulence computation using the FLASH3 application framework on the IBM Blue Gene/L system," *IBM Journal of Research and Development*, vol. 52, pp. 127-136, 2008.

[3] A. Acharya, M. Uysal, and J. Saltz, "Active disks: Programming model, algorithms and evaluation," in *ACM SIGPLAN Notices*, 1998, pp. 81-91.

[4] J. Dongarra et al., "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, pp. 3-60, 2011.

[5] N. Abbani et al., "A distributed reconfigurable active SSD platform for data intensive applications," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, 2011, pp. 25-34.

[6] M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage: pushing more functionality into storage," *Potentials, IEEE*, vol. 24, pp. 31-34, 2005.

[7] P. Francisco, "The netezza data appliance architecture: A platform for high performance data warehousing and analytics," IBM, Tech. Rep, 2011.

[8] F. Dong and S. G Akl, "Scheduling algorithms for grid computing: State of the art and open problems," Technical Report, 2006.

[9] P. Visalakshi and SN. 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.

[10] J. Xu, A. YS Lam, and V. OK Li, "Chemical reaction optimization for task scheduling in grid computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 10, pp. 1624-1631, 2011.

[11] S. Guo, H. Huang, Z. Wang, and M. Xie, "Grid service reliability modeling and optimal task scheduling considering fault recovery," *Reliability, IEEE Transactions on*, vol. 60, no. 1, pp. 263-274, 2011.

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

[13] 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*, 2012, pp. 654-657.

[14] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, p. 85, 2012.

[15] PK Yadav, MP Singh, and K. Sharma, "An Optimal Task Allocation Model for System Cost Analysis in Heterogeneous Distributed Computing Systems: A Heuristic Approach," in *International Journal of Computer Applications*, vol. 28, 2011, pp. 30-37.

[16] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci, "Task assignment in heterogeneous computing systems," *Journal of parallel and Distributed Computing*, vol. 66, no. 1, pp. 32-46, 2006.

[17] Q. Kang, H. He, and H. Song, "Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm," *Journal of Systems and Software*, vol. 84, no. 6, pp. 985-992, 2011.

[18] G. Attiya and Y. Hamam, "Task allocation for minimizing programs completion time in multicomputer systems," in *Computational Science and Its Applications--ICCSA 2004*, 2004, pp. 97-106.

[19] Q. S. Hua, Z. G. Chen, and F. C. Lau, "A new method for independent task scheduling in nonlinearly DAG clustering," in *Parallel Architectures, Algorithms and Networks, 2004. Proceedings. 7th International Symposium on*, 2004, pp. 187-192.

[20] N. Arora, "Analysis And Performance Comparison Of Algorithms For Scheduling Directed Task Graphs To Parallel Processors," *ANALYSIS*, vol. 4, p. 2, 2012.

[21] M. Katsev, J. Yu, and S. M LaValle, "Efficient Formation Path Planning on Large Graphs," in *IEEE International Conference on Robotics and*

Automation (ICRA), 2013.

[22] S. Frey and T. Ertl, "PaTraCo: a framework enabling the transparent and efficient programming of heterogeneous compute networks," in *Proceedings of the 10th Eurographics conference on Parallel Graphics and Visualization*, 2010, pp. 131-140.

[23] R. Bellman, "On a routing problem," DTIC Document, 1956.

[24] T. Rauber and G. Runger, *Parallel programming: For multicore and cluster systems.*: Springer-Verlag New York Incorporated, 2010.

[25] M. Ahmed, A. S. M. R. Chowdhury, M. Ahmed, and M. M. H. Rafee, "An advanced survey on cloud computing and state-of-the-art research issues," *International Journal of Computer Science Issues(IJCSI)*, vol. 9, p. 1, 2012.

[26] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7-18, 2010.

[27] Amazon EC2. [Online]. http://aws.amazon.com/ec2/

[28] Microsoft Windows Azure Platform. [Online]. http://www.windowsazure.com/

[29] Google App Engine. [Online]. https://developers.google.com/appengine/

[30] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107-113, 2008.

[31] M. Awad, "FPGA supercomputing platforms: a survey," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 564-568.

[32] T. El-Ghazawi and K. Gaj, "Reconfigurable supercomputing systems," *Proc. tutorial presented during Supercomputing*, 2004.

[33] M. C. Smith, J. S. Vetter, and X. Liang, "Accelerating scientific applications with the SRC-6 reconfigurable computer: Methodologies and analysis," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, p. 157.

[34] S. Hauck and A. DeHon, *Reconfigurable computing: the theory and practice of FPGA-based computation.*: Morgan Kaufmann, 2010.

[35] S. W. Son et al., "Enabling active storage on parallel I/O software stacks," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1-12.

[36] Y. Chen, C. Chen, X. Sun, W.D. Gropp, and R. Thakur, "A decoupled

execution paradigm for data-intensive high-end computing," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, 2012, pp. 200-208.

[37] F. Zheng et al., "PreDatA-preparatory data analytics on peta-scale machines," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1-12.

[38] H.Z. Jia, Y.C. Nee, Y.H. Fuh, and Y.F. Zhang, "A modified genetic algorithm for distributed scheduling problems," *Journal of Intelligent Manufacturing*, vol. 14, pp. 351-362, 2003.

[39] T. Davidovic and T. Crainic, "Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems," *Computers & operations research*, vol. 33, pp. 2155--2177, 2006.

[40] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, pp. 379-394, 2002.

[41] IBM ILOG CPLEX. (Available) IBM. [Online]. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/

[42] C. E. Leiserson, "The Cilk++ concurrency platform," *The Journal of Supercomputing*, vol. 51, no. 3, pp. 244-257, 2010.