TOPOLOGY-AWARE OPTIMAL TASK ALLOCATION FRAMEWORK FOR MISSION CRITICAL ENVIRONMENT: CENTRALIZED AND DECENTRALIZED APPROACHES

BY

SHAMEEM AHMED

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Advisor:

Professor Klara Nahrstedt

ABSTRACT

A Mission Critical Environment (MCE) consists of error-prone, highly variable, and highly rate limited communication channels. Paradoxically, this environment substantially increases the need to perform Optimal Task Allocation (OTA), while at the same time making it much harder to perform OTA efficiently. To perform OTA in MCE, in this thesis, I have proposed two novel automated algorithms. The first algorithm is called Centralized Optimal Task Allocation Algorithm (COTAA), where I consider OTA for publish/subscribe-based MCE since it has unique characteristics such as high level publish/subscribe node and task differentiation and high scalability. I also propose an architectural framework and communication protocols emphasizing the unique challenges of MCE. I adopt well known *Hungarian Algorithm* and *Rectangular Assignment Algorithm* to solve the OTA problem in polynomial time. The second algorithm is called Decentralized Optimal Task Allocation Algorithm (DOTAA) which exploits the concept of application-layer Distributed Hash Table (DHT) to perform OTA in MCE. Through simulations, I evaluate the performance of both COTAA and DOTAA for multiple mission critical scenarios. The results indicate that both COTAA and DOTAA achieve the goal of OTA in highly dynamic MCEs, with low processing time and communication overhead.

ACKNOWLEDGMENTS

I would cordially like to thank my advisor Professor Klara Nahrstedt for her invaluable time and proper guidance over the last few years. I would also like to thank Boeing Research and Technology for sponsoring this thesis. I am thankful to Dr. Thadpong Pongthawornkamol, Professor Matthew Caesar and Dr. Guijun Wang for their valuable suggestions to improve the quality of my research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Mission Critical Environments (MCEs), such as battlefields, emergency response, firefighting, and rescue operations, are among the most challenging operating environments for distributed protocols. These environments present several of the most extreme systems-level design challenges, including high rates of mobility and dynamism, communication channels prone to high rates of errors and loss, and unpredictable and highly variable network delays and bandwidths [7]. And yet MCEs also represent a situation where demands on the correct behavior and efficient performance of systems are at their highest - with lives hanging in the balance, tolerance for performance problems, lost communications, and unpredictable performance is extremely low. Figure 1.1 offers an example of such MCE which shows a post-disastrous situation in a particular geographic location. To cope with these demanding environments, mission operators run systems to perform Optimal Task Allocation (OTA). These systems are often comprised of a set of distributed nodes (e.g., communication devices carried by soldiers in the battlefield), which are generally equipped with data processing, memory, and communication capabilities. Each node may be responsible for carrying out multiple tasks (e.g., sensing the temperature and taking pictures using camera).

OTA methods have proven useful in other contexts, including dynamic trajectory planning in robotics [5, 6, 9], resource allocation in sensor networks [11, 12, 13], and grid computing [10]. However, performing OTA in MCEs presents new challenges, as well as new opportunities to customize their design and execution for the settings I consider. For example, MCEs often maintain a logically-centralized command and control node which orchestrates the operation, eliminating the need for fully distributed techniques. At the same time, MCEs

Figure 1.1. An Example of Mission Critical Environment (MCE)

introduce new challenges, including the need to react in real-time to events, and the need to operate in resource constrained and highly variable-performance environments. Currently, in MCEs, users often rely on *manual* allocation - a human at a *command and control center* is responsible for allocating the tasks to the nodes. Unfortunately, relying on human users to perform this allocation during disaster situations leads to potential for mistakes and inefficient task allocations. There are only few existing solutions [32, 33] which perform OTA automatically. However, these solutions suffer from several problems such as:

1. Existing solutions are applicable only for very specific MCE such as slower UAV (Unmanned Aerial Vehicle) operation.
2. Existing solutions are not scalable and only applicable for very small settings.
3. Existing solutions perform very poorly when the nodes move very fast.
4. The convergence of the existing solutions is very slow.
5. Existing solutions don’t always guarantee the optimality of the task allocation.

1.1. Problem Description

A mission consists of several tasks where each task requires a set of resources and adheres to system constraints. In this thesis, I consider task allocation as an optimization problem. The objective of the optimization is to maximize the system's overall utility. This utility value mainly depends on the existing resources of the nodes as well as the importance of those resources for a particular mission. For this purpose, it is utmost important to precisely define a utility / cost function to measure the system utility. Besides defining the utility function, the following research challenges are need to be addressed:

1. Design a communication protocol to know the resource status of all the nodes eligible to perform a task.
2. Design an automated, online, time-efficient, and optimal algorithm to perform task allocation. An offline algorithm might not fit well for MCEs since the status of the system (e.g., geographical positions of the nodes, resource status of the nodes, etc.) change rapidly. The algorithm should converge very quickly; otherwise, tasks might not be executed before its deadline. The algorithm should be optimal to ensure that no task is left unassigned and the system's utility value is maximized.
3. Design a communication protocol for task assignment which informs the best eligible node to start executing the task.
4. Make the entire system topology-aware to ensure that task assignment can be updated due to topology change (e.g., node mobility, change of the nodes' resource status, etc.).
5. Make the entire system scalable so that it works for large number of nodes and tasks.
6. Make the entire system generalizable so that it can be applied to a large group of MCEs.

1.2. Contributions of the Thesis

In this thesis, to address the research challenges described in Section 1.1, I have developed two automated online approaches namely Centralized Optimal Task Allocation Algorithm (COTAA) and Decentralized Optimal Task Allocation Algorithm (DOTAA). These two approaches are scalable, topology-aware, fast, and guarantee the optimality of task allocation. Moreover, the solutions are applicable for a broad range of MCEs such as battlefields, emergency response, firefighting, rescue operations, and UAV operation, etc.

COTAA has three major contributions:

1. I map publish/subscribe communication model to OTA problem because publish/subscribe model is proven useful for several MCEs [37]. The mapping considers a formulation in which tasks are performed and distributed across nodes in a manner that maximizes the aggregate network-wide utility.
2. I provide an online algorithm, protocol, and network architecture for carrying out this task allocation, given an arriving set of tasks as inputs. This algorithm addresses the challenges of OTA in MCEs described earlier. In particular, I
    a. Develop a technique to identify availability of nodes and their resources. This technique is required to understand the eligibility of a node to perform a particular task. Moreover, if there are several eligible nodes to perform a single task, this technique helps to find the best eligible node for that task.
    b. Exploit the *Hungarian Algorithm* and the *Rectangular Assignment Algorithm* to find OTA solutions. Both of these algorithms are proven effective to perform OTA under resource constraint conditions.

c. Assign tasks to the appropriate nodes via information dissemination protocol. As MCE poses DIL (Disconnected, Intermittent, Limited) communication among nodes, this protocol assigns the tasks to the best eligible node in an efficient way.
d. Update the network resource allocation and task re-assignment based on the current position and mobility of the nodes. In some MCEs (e.g., UAV operation), nodes move very fast. Hence, task re-assignment might be required.

3. I perform extensive performance evaluation via simulations for several publish/subscribe scenarios.

Some MCEs don't have any central unit. All the nodes need to take decision by its own or by collaborating with the other nearby nodes. So it is required to have another solution where the research challenges can be addressed in a distributed manner. DOTAA is used to serve this purpose. DOTAA has three major contributions:

1. I propose an automated, online, completely decentralized algorithm along with new network protocol and architecture fitted for any kind of MCEs.
2. I exploit the *Distributed Hash Table (DHT)*-based approach to solve the OTA problem, which, to the best of our knowledge, is the very first effort, in this context.
3. I extensively evaluate the performance of DOTAA through ns-2 simulator for several mission critical scenarios.

1.3. Thesis Outline

The outline of this thesis is as follows. Chapter 2 contains the background information. Both chapter 3 and chapter 4 present the research challenges and the proposed approaches for OTA in MCEs. While chapter 3 concentrates on the centralized approach (COTAA), chapter 4 focuses

on the decentralized approach (DOTAA). I give a detailed description of my simulation implementation in chapter 5. Chapter 6 details the experiments I have run for COTAA and DOTAA along with the evaluation results. Chapter 7 describes the current state of the art. In chapter 8, I conclude with some novel research directions of future work.

CHAPTER 2: BACKGROUND

This chapter describes all the basic terms used in this thesis to make those comprehensible to the readers.

Figure 2.1. Haiti Earthquake [24]

Figure 2.2. Disaster in Japan [25]

Figure 2.3. Fire fighters in action [26]

Figure 2.4. Post-Disaster Rescue [27]

2.1. Mission Critical Environment (MCE)

By MCE, I mean the environment where a mission needs to be achieved in a timely-manner. A mission consists of several tasks which have specific start and end times and which are constrained by specific resource parameters. So, a mission is said to be successfully completed if all tasks of that mission are performed before the deadlines and adhere to the resource requirements. Figures 2.1 and 2.2 present two disaster scenarios (e.g., Haiti Earthquake and

Japan Disaster) where a mission is defined as how to rescue the affected people. This mission is critical in a sense that affected people need to be rescued as quickly as possible; otherwise the casualty might be enormous. Figures 2.3 and 2.4 show two rescue operations. In Figure 2.3, firefighters are in action to control the fire, while Figure 2.4 shows how members of the rescue team help the affected people in a post-disaster situation.

2.2. Task Allocation (TA) and Optimal Task Allocation (OTA)

Task allocation is a process where particular workers (eligible to perform the tasks) execute one or more tasks assigned to them [36]. If there are several workers and tasks and there are multiple ways to perform all the tasks by all the workers, then Optimal Task Allocation (OTA) ensures that:

- There is no unassigned task after the task allocation (*TA*).
- The utility of *TA* is maximum in a sense that any other task allocation can't guarantee more system utilization than *TA*.

2.3. Publish/Subscribe System

Publish/Subscribe System [3, 4, 24] provides a communication pattern among two types of nodes:

1. Publisher node and
2. Subscriber node

A publisher node publishes a message and a subscriber which subscribes for that message receives it from the publisher. The entire communication occurs in an asynchronous manner. Based on message filtering, there are three types of publish/subscribe systems:

1. **Topic-based publish/subscribe system:** A publisher node publishes a message with a topic (or a set of topics) and the subscriber node subscribing to that topic receives the message.
2. **Content-based publish/subscribe system:** The subscriber defines some constraints in its subscription process. As soon as the publisher publishes a message and the message content adheres to the constraints imposed by the subscriber, the message is delivered from the publisher to the subscriber.
3. **Hybrid approach:** In this case, publisher follows the topic-based approach by publishing a message based on a specific topic. However, the subscriber follows the content-based approach where the subscriber not only subscribes for a specific topic but also specifies the constraints for that topic. As soon as both the topic and the constraints match, the message is delivered from the publisher to the subscriber.

The main advantages of using a publish/subscribe system are as follows [23]:

1. Publishers and subscribers are loosely coupled. They don't need to know the existence of each other. It differs from traditional client/server communication model. In a client/server model, both client and server must be up at the same time for any communication. However, publishers and subscribers don't need to be up at the same time.
2. Publish/subscribe communication system is very scalable.

CHAPTER 3: CENTRALIZED OPTIMAL TASK ALLOCATION ALGORITHM (COTAA)

To solve the OTA problem in MCE, I have first proposed a centralized approach called Centralized Optimal Task Allocation Algorithm (COTAA).

3.1. Models and Assumptions

Table 3.1 presents the notations I use to describe COTAA.

3.1.1. Network Model

I model the network as an infrastructure based wireless network. The network is considered as an undirected graph, *G(V, E)*, where *V* represents the set of nodes and *E* represents the set of communication links or edges. *V* can also be classified into four sub-categories namely V_{pub} (set of publisher nodes), V_{sub} (set of subscriber nodes), *B* (set of InfoBrokers), and *CU* (Central Unit), where

$$V = V_{pub} \cup V_{sub} \cup B \cup CU$$

$$N = V_{pub} \cup V_{sub}$$

E can be classified into three sub-categories namely E_{PB} (an edge between publisher and InfoBroker), E_{SB} (an edge between subscribe and InfoBroker), and E_{BC} (an edge between InfoBroker and Central Unit), where

$$E = E_{PB} \cup E_{SB} \cup E_{BC}$$

Table 3.1. Notation Table for COTAA

Notation	Meaning
N	Set of Publisher or Subscriber Nodes
$n = \|N\|$	Number of Publisher or Subscriber Nodes
T	Set of Tasks
T_{pub}	Set of Publisher Tasks
T_{sub}	Set of Subscriber Tasks
$m = \|T\|$	Number of Tasks
i	Node identifier
τ_j	Task identifier
τ_{ij}	Task τ_j assigned to node *i*
$\tau_{ij}(t)$	Task τ_j assigned to node *i* at time *t*
η_i	Number of tasks assigned to node *i*
$\eta_i(t)$	Number of tasks assigned to node *i* at time *t*
P	Set of Resource Parameters
$p = \|P\|$	Number of Resource Parameters
U_{ij}	Utility Value of Node *i* for Task τ_j
W_{jk}	Weight for Resource Parameter *k* of Task τ_j
P_{ik}	Resource Parameter *k* of Node *i*
α_{ij}	Node *i* is eligible for Task τ_j
λ_i	Node status for Node *i*
PT_i	Set of Topics that Node *i* can Publish
ST_i	Set of Topics that Node *i* can Subscribe
Θ	A single topic for Publish/Subscribe Model

3.1.2. Node Model

Each node (*i*) in the network model is assigned a unique identifier and computes its node status (λ_i). To compute λ_i, node (*i*) needs to know the following information:

- The status of each resource parameter node (*i*) posses. CPU, available memory, bandwidth, communication delay, etc. are few examples of the resource parameters.
- The set of topics node (*i*) can publish (PT_i)
- The set of topics node (*i*) can subscribe to (ST_i), and
- The list of tasks node (*i*) is currently executing.

Each node can move across the domains arbitrarily. Security aspect of node model is outside of the scope of this thesis. However, existing techniques such as key encryptions and node certificates can be used to address the security issues.

3.1.3. Publish/Subscribe Model

Each node (*i*) can carry either a publisher task or a subscriber task or both. The publisher and subscriber nodes are grouped into domains according to various criteria such as geographic location, communication range, and size of the groups. Each domain is managed by an *InfoBroker*. The *InfoBroker* communicates with each node, exchanging publisher and subscriber information.

3.1.4. Task Model

Tasks are classified into two categories: publisher task (T_{pub}) and subscriber task (T_{sub}), where $T = T_{pub} \cup T_{sub}$. For example, a node equipped with temperature sensor can publish temperature on the network. Such a task is considered as a publisher task. On the contrary, a subscriber node can receive the temperature value from the publisher via the *InfoBroker*. This task is an example of a

subscriber task. A task set (T) is provided as a mission to the *CU* (Central Unit) from the corresponding authority (e.g., commander-in-charge) and COTAA assigns the tasks to the appropriate nodes optimally. Each task (τ_j) has few properties, namely type, topic, a list of available resource parameters, and corresponding weight of those resource parameters. The type field contains either 'pub' or 'sub' value. Topic field specifies in which topic task (τ_j) belongs to. The resource parameter (P_{ik}, $\forall k \in P$) of each node (i) is prioritized based on the task weight (W_{jk}, $\forall k \in P$). W_{jk} is application and task-specific, and only the commander-in-chief knows the relative importance of a resource for a particular task. Hence, W_{jk} is assigned by the commander-in-chief.

A node (i) can perform several tasks. Eligibility (α_{ij}) of a task (τ_j) for node (i) can be considered as an admission-control problem. Existing research [2] addresses the admission-control problem, hence, it is not considered in this thesis. Task eligibility (α_{ij}) is defined as if node (i) has sufficient resources to execute task (τ_j) and also adheres to the following condition:

$$\alpha_{ij} = \begin{cases} 1 \quad iff \ ((\tau_j.type == pub \ AND \ \tau_j.topic \ \epsilon \ PT_i) \ OR \ (\tau_j.type == sub \ AND \ \tau_j.topic \ \epsilon \ ST_i) \\ 0 \quad otherwise \end{cases}$$

3.1.5. Assumptions

In COTAA, the following assumptions have been made:

1. Mobile nodes have to follow the communication pattern of a publish/subscribe-based hierarchical communication system where each mobile node (e.g., publisher or subscriber) only communicates to the broker nodes. The broker works as the bridge between the mobile nodes and the central node.
2. There is no inter-dependency among tasks of a mission.
3. The central unit is solely responsible for the task allocation.

3.2. Topology

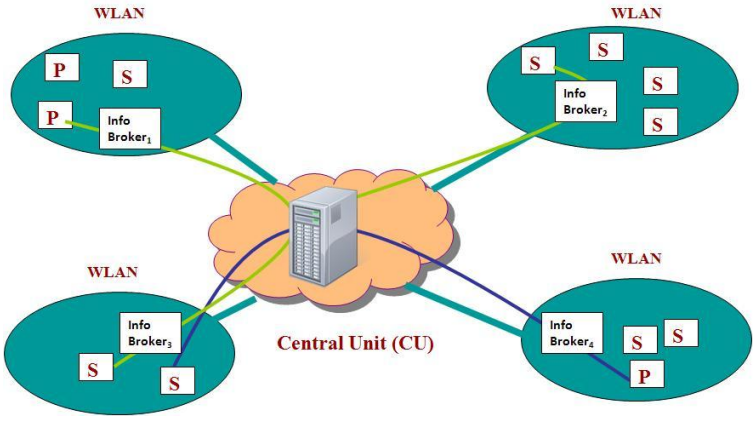


Figure 3.1. Publish/Subscribe-based Mission Critical Environment

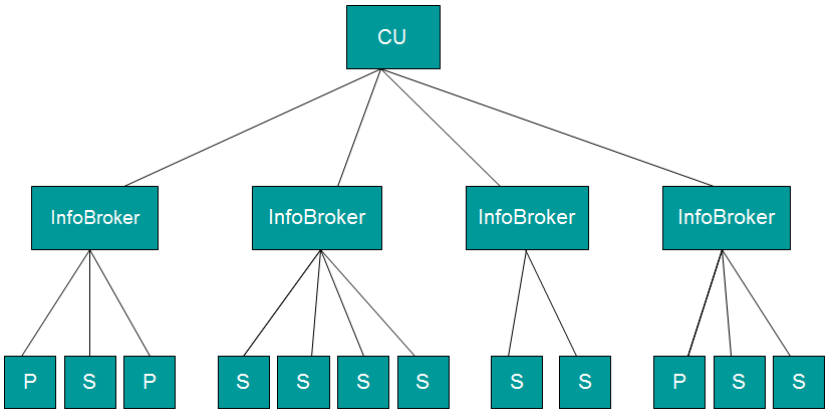


Figure 3.2. Hierarchical Publish/Subscribe-based Structure

COTAA exploits the publish/subscribe model to use it in MCE. Due to highly dynamic nature of MCEs, an individual node has little to no knowledge about the other nodes. As a result, nodes rely on the broker for information dissemination and as the broker is aware of the needs of the existing nodes, it can handle the distribution efficiently. Figure 3.1 presents the proposed COTAA architecture where each *Wireless Local Area Network (WLAN)* represents a domain where a publisher or subscriber can reside. Any publisher or subscriber node is only allowed to

communicate to their corresponding gateway, which is called *InfoBroker*. The *InfoBroker*, on behalf of the publisher or the subscriber, communicates to the other domains' *InfoBroker* through Central Unit (*CU*). Figure 3.2 presents the logical architecture of such a publish/subscribe-based MCE which follows a hierarchical network structure.

3.3. Problem Statement

3.3.1. Problem Description

The research challenges of COTAA are as follows:

1. Design an information dissemination protocol so that each node (*i*) informs *CU* about its resource status (λ_i) and task eligibility ($\forall j\ \alpha_{ij}$);
2. Find time-efficient and optimal task allocation under resource constraint conditions;
3. Design an efficient task assignment protocol allowing *CU* to assign tasks (τ_j) to appropriate nodes (*i*) efficiently;
4. Make the entire system topology-aware so that the task assignment can be updated based on dynamic change of the topology (e.g., node mobility, change of the node's resource status, etc.).

Publish/subscribe-based MCE demands to execute OTA in polynomial time. For COTAA, I consider three categories of OTA based on the number of tasks (*m*) and number of nodes (*n*): (a) $n = m$, (b) $n > m$, and (c) $n < m$. I adopt the *Hungarian Algorithm* [8] and the *Rectangular Assignment Algorithm* [1] to solve the first and second categories respectively. For the third category, I apply an *iterative approach*. I divide the third category problem into $\left\lfloor \frac{m}{n} \right\rfloor$ iterations

of the first category problem ($n = m$). At the last iteration, the number of tasks is $(m - \left\lfloor \frac{m}{n} \right\rfloor * n)$, which is less than n. Hence, I use the *Rectangular Assignment Algorithm* for the last iteration.

3.3.2. Problem Formulation

A critical mission is a set of tasks where each task requires a set of resources and needs to consider system constraints. I formulate task allocation among nodes as an *optimization problem*. The objective of this optimization is to maximize the overall utility value of the system. This utility value largely depends on the existing resources of the nodes as well as how important those resources are for a particular task or mission. For node (i) and task (τ_j), I use the following utility function:

$$U_{ij} = \begin{cases} \sum_{k=1}^{p} \frac{P_{ik}}{\max(P_{ik})} \times W_{jk} \;\; iff \;\; (i \in N \; AND \; \alpha_{ij} = 1) \\ -\infty \;\; otherwise \end{cases}$$

Each P_{ik} (resource parameter k of node i) has different measurement unit such as CPU processing time in sec, available memory in bytes, communication delay in sec and bandwidth in bps. Here, $\max(P_{ik})$ is used to normalize P_{ik} and makes U_{ij} unit-less. For example, the available memory and CPU processing time should be normalized by dividing these with the maximum available memory and CPU processing time, respectively.

The objective function is targeted to maximize the total utility value of all nodes and all tasks:

$$\max(U) = max_{i \in N \; AND \; \alpha_{ij}} \left[\sum_{j=1}^{m} U_{ij} \right]$$

Table 3.2. Example of a Node-Task Matrix

	τ_1	τ_2	τ_3
N_{11}	U_{11}	U_{12}	U_{13}
N_{12}	U_{21}	U_{22}	$-\infty$
N_{32}	$-\infty$	U_{32}	U_{33}
N_{43}	U_{41}	U_{42}	U_{43}

Suppose, there are four nodes (N_{11}, N_{12}, N_{32}, and N_{43}) and three tasks (τ_1, τ_2, and τ_3). Table 3.2 presents the corresponding utility parameters for this particular setting. In Table 3.2, a value of $-\infty$ indicates that the node is not eligible for that particular task. The total utility value would be as follows:

$$\max(U) = \max(U_{11} + U_{22} + U_{33}\,, U_{11} + U_{22} + U_{43}, U_{11} + U_{32} + U_{43}, \ldots\ldots)$$

In addition to maximizing the utility value, *T* must be successfully allocated to *N*, provided there are sufficient nodes which are eligible for executing *T*.

$$T = \begin{cases} \bigcup_{j=1}^{m} \tau_{ij} & n \geq m \\ \bigcup_{j=1}^{n} \tau_{ij}(t_1) \bigcup_{j=n+1}^{2n} \tau_{ij}(t_2) \ \ldots\ldots\ldots \bigcup_{j=m-n+1}^{m} \tau_{ij}\left(t_{\frac{m}{n}}\right) & n < m \end{cases}$$

The above optimization is subject to the following constraints:

$$i \in N, \qquad \tau_{ij} \in T, \qquad \tau_{ij}(t) \in T, \quad \eta_i \geq 0, \qquad and \quad t_1 < t_2 < t_3 \ldots\ldots$$

3.4. Proposed Solution

3.4.1. Information Dissemination Protocol

Each publisher and subscriber node (*i*) computes its node status (λ_i). Each node (*i*) periodically sends a beacon to its corresponding *InfoBroker* along with λ_i piggy-backed. Each *InfoBroker* periodically aggregates all λ_i and stores these in a matrix, called *Node-Resource Matrix (NRM)*. The *InfoBrokers* communicate to *CU* to submit their *NRMs*. With the information from the *InfoBrokers*, *CU* generates its own matrix, termed *Node-Task Matrix (NTM)* by aggregating all *NRMs*. Each entry of *NTM* is either a singular value generated from the utility function or $-\infty$. Algorithms 1, 2, and 3 present the pseudo-code of *NRM* Generation, *NTM* Generation, and OTA respectively. Each *InfoBroker* runs the *GenerateNRM Algorithm* while *CU* executes the remaining two.

The *Hungarian Method* [8], one of the most celebrated algorithms in combinatorial optimization area, solves the assignment problem in polynomial time. This algorithm assumes that the number of nodes (*n*) is equal to the number of tasks (*m*). The time complexity of this method is $O(n^3)$. The *Rectangular Assignment algorithm* [1] solves the generalized assignment problem where the number of nodes (*n*) can be larger than the number of tasks (*m*). The time complexity of this method is $O(n^2m)$. Note that the *Hungarian Algorithm* and the *Rectangular Assignment Algorithm* are mainly designed for minimization rather than maximization. To address this issue, for implementation purpose, I subtract each utility value (U_{ij}) from the maximum of all utility values.

Algorithm 1 GenerateNRM ()

b=current InfoBroker
NRM_b = Ø /* Initialize *NRM* for *b* */
for each node *i* assigned to *b* **do**
compute node status λ_i
generate an entry at NRM_b for λ_i
end for
Send NRM_b to *CU*

Algorithm 2 GenerateNTM (*N, T, B*)

NTM = Ø /* Initialize NTM */
for each InfoBroker $b \in B$ **do**
get NRM_b
for each node *i* assigned to *b* **do**
for each task $j \in T$ **do**
calculate U_{ij}
$NTM_{ij} = U_{ij}$
end for
end for
end for

Algorithm 3 OTA (*N, T, B*)

NTM = GenerateNTM (*N, T, B)*
NTAM = Ø /*Initialize *NTAM* */
complete = false
while (*complete==false*) **do**
if (*m==n*) **then**
NTAM = NTAM $\cup$ *Hungarian (n, N, T)*
complete = true
else if (*m<n*) **then**
NTAM = NTAM $\cup$ *Rectangular Assignment (m, n, N, T)*
complete = true
else
T_f = first *n* tasks of *T*
NTAM = NTAM $\cup$ *Hungarian (n, N,* T_f*)*
m = m – n
$T = T \setminus T_f$
NTM = GenerateNTM (*N, T, B)*
end if
end while

3.4.2. Example

Suppose four nodes (N_{11}, N_{12}, N_{13}, and N_{14}) are connected to *InfoBroker*$_1$ and five nodes (N_{21}, N_{22}, N_{23}, N_{24}, and N_{25}) are connected to *InfoBroker*$_2$. Table 3.3 and Table 3.4 show the *NRM* maintained by two *InfoBrokers*. Table 3.5 shows the task list (τ_1, τ_2, and τ_3) and the weight of the corresponding resource parameters. Higher value of weight indicates the higher preference of the resource parameters. *CU*, by using Algorithm 2, generates *NTM* which is shown in Table 3.6. By subtracting each element of *NTM* from the maximum value, 27.93 (from Table 3.6) and then applying the *Rectangular Assignment Algorithm* [1], the OTA would be as follows: T_1 is assigned to N_{13}, T_2 is assigned to N_{22}, and T_3 is assigned to N_{25}, which is shown in Table 3.7.

Table 3.3. Node-Resource Matrix (*NRM*) at *InfoBroker*$_1$

	CPU	*Mem*	*BW*	*Delay*	*PTi*	*STi*	*CurrTask*
N_{11}	12	100	134	10	Θ_1 , Θ_2	Θ_4 , Θ_7	τ_7
N_{12}	23	150	74	20	Θ_1 , Θ_2	Θ_8	X
N_{13}	24	300	23	30	Θ_1 , Θ_2	Θ_3	X
N_{14}	48	230	56	15	Θ_{12}	Θ_{13}	τ_{13}

Table 3.4. Node-Resource Matrix (*NRM*) at *InfoBroker*$_2$

	CPU	*Mem*	*BW*	*Delay*	*PTi*	*STi*	*CurrTask*
N_{21}	134	28	453	5	Θ_{12}	Θ_3 , Θ_6	τ_{12}
N_{22}	876	500	125	31	Θ_1 , Θ_2	Θ_8	X
N_{23}	98	456	765	34	Θ_7	Θ_{10}	X
N_{24}	14	234	541	17	X	Θ_{13}	τ_{13}
N_{25}	87	987	876	20	X	Θ_3 , Θ_{13}	X

Table 3.5. Requested Task List at Central Unit

	Type	*Topic*	*CPU W.*	*Mem W.*	*BW W.*	*Delay W.*
τ_1	*pub*	Θ_1	*3*	*9*	*15*	*20*
τ_2	*pub*	Θ_2	*10*	*5*	*3*	*2*
τ_3	*sub*	Θ_3	*30*	*8*	*1*	*1*

Table 3.6. Node-Task Matrix *(NTM*) at Central Unit

	τ_1	τ_2	τ_3
N_{11}	*9.13*	*1.69*	$-\infty$
N_{12}	*14.48*	*2.45*	$-\infty$
N_{13}	*20.86*	*3.64*	*4.16*
N_{21}	$-\infty$	$-\infty$	*5.48*
N_{22}	*27.93*	*14.78*	$-\infty$
N_{25}	$-\infty$	$-\infty$	*12.57*

Table 3.7. Node-Task Assignment Matrix (*NTAM*) at Central Unit

	τ_1	τ_2	τ_3
N_{11}	*0*	*0*	*0*
N_{12}	*0*	*0*	*0*
N_{13}	*1*	*0*	*0*
N_{21}	*0*	*0*	*0*
N_{22}	*0*	*1*	*0*
N_{25}	*0*	*0*	*1*

3.4.3. Task Assignment Protocol

After executing Algorithm 3 (*OTA(N,T,B)*), *CU* assigns the tasks to the appropriate nodes via the corresponding *InfoBrokers*. However, due to the lossy nature of the wireless medium, the task assignment message sent from an *InfoBroker* might be lost. To ensure eventual task assignment, I adopt the reactive mechanism as follows. Each node (*i*) periodically sends the node status message (λ_i) to the *InfoBroker*. If λ_i does not contain the task recently assigned to that node, the *InfoBroker* can infer that the task assignment message sent to that node was lost. In situations like these, the *InfoBroker* re-sends the task assignment message to that node.

3.4.4. Topology-Aware Protocol

The topology of the network can be changed due to the node mobility and change of the node status. To achieve topology awareness, I assume the task re-allocation model as follows. Whenever any node moves out from its closest *InfoBroker*, all tasks that are running on that node must be revoked and re-allocated. The re-allocation process is triggered at *CU* whenever it detects such a change from the *NRMs* aggregated from all *InfoBrokers*. However, to prevent *CU* from re-allocating tasks too frequently, *CU* only checks for lost tasks and performs the corresponding re-allocation periodically.

CHAPTER 4: DECENTRALIZED OPTIMAL TASK ALLOCATION ALGORITHM (DOTAA)

To solve the OTA problem in MCE, I have proposed a decentralized approach called Decentralized Optimal Task Allocation Algorithm (DOTAA).

4.1. Topology

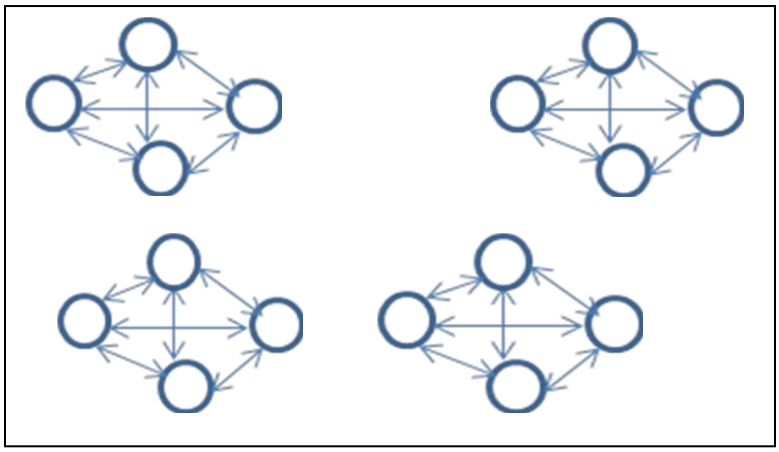


Figure 4.1. Decentralized Optimal Task Allocation Algorithm (DOTAA) Topology

The topology used in DOTAA differs from the topology used in COTAA. In case of DOTAA, each group of nodes forms a full mesh topology (Figure 4.1) and each node is single-hop away from all other nodes of the same group.

4.2. Models and Assumptions

Table 4.1 presents the notations I use to describe DOTAA.

4.2.1. Network Model

Here, I consider the network model as (a) static wireless ad-hoc network and (b) mobile wireless ad-hoc network. Each node can communicate to other nodes only through wireless communication provided the other nodes are within its wireless communication range.

Table 4.1. Notation Table for DOTAA

Notation	Meaning
T	Set of Tasks
R	Resource requirement to perform a task
i	Node identifier
τ	Task identifier
M	A Mission
HashNode	A node which evaluates all the cost values and then selects a node to perform a task
C	Cost value a node calculates and then sends to *HashNode*

4.2.2. Node Model

Each node is a mobile node equipped with different resources (e.g., sensors) required to perform a task. Each node has a unique identifier and maintains a list of its available resources and a list of currently executing tasks.

4.2.3. Mission and Task Model

A deadline-driven mission consists of several tasks which are of two types: *Computation-intensive tasks* and *Communication-intensive tasks*. Each task has a unique identifier, start and end time, and resource requirement to perform that task. A node is eligible to perform a task if and only if that node has sufficient resources required for that particular task. There is a correlation between a node and a task identifier. A hash value of a task identifier always provides the identifier of a particular node.

4.2.4. Assumptions

The assumptions applied to COTAA are relaxed for DOTAA. For instance, no publish/subscribe-based communication pattern is considered in DOTAA. Also, there is no central unit for resource collection and task allocation. Unlike COTAA, there might be *task-dependency*, i.e., a task (τ_1) can start if and only if another task (τ_2) finishes first. However, few additional assumptions have been made for DOTAA, such as:

1. Each node belonging to a group is single-hop away from other nodes of the same group.
2. No node fails during the mission.
3. Each node is an honest node. For instance, when a node calculates its bid value, it correctly calculates it. In case of *arbiter node* (described later), it evaluates bid value correctly without showing any bias towards any particular node.

4.3. Problem Description

DOTAA addresses the following research challenges:

1. To perform OTA, it is utmost important to know the resource status of all nodes who are eligible to perform a particular task. In the centralized approach, the resource status of each node is periodically reported to the Central Unit, *CU*. However, for distributed system, there is no such central unit. Hence, it is difficult to get the resource status in a very efficient manner. The first research challenge of DOTAA is to design a distributed networking protocol so that the resource status of eligible nodes can be collected in an efficient manner.
2. After getting the resource status, it is required to evaluate all the eligible nodes so that the best node can be chosen for a particular task. The second challenge of DOTAA is to find

such an efficient evaluation method. Also, a distributed protocol is necessary to inform the best eligible node to start executing the task.

3. The final challenge of DOTAA is to make the proposed distributed protocols topology-aware in the sense that even when the node moves with great speed, DOTAA finds the best node to perform the task optimally.

4.4. Proposed Solution

During the mission, each node executes DOTAA which consists of three major algorithms:

- *Algorithm 1: AdmissionControl Algorithm*
- *Algorithm 2: MissionAccomplishment Algorithm*
- *Algorithm 3: NodeSelectionForTask Algorithm*

AdmissionControl Algorithm ensures a node's eligibility to perform a task. Each task (τ_j) has resource requirements which an eligible node must support. For instance, if a task is to take a picture of a particular geographic location periodically, the eligible node must be equipped with a camera; if a task needs to publish the temperature, the eligible node must be equipped with a temperature sensor, and so on. According to the *AdmissionControl Algorithm*, a node ensures its eligibility for a task by checking whether it has sufficient resources required for that task. The eligible node allocates the resources required for that task. Later, if the node is not assigned to the task, it releases those resources.

MissionAccomplishment Algorithm is executed by each node locally to accomplish a particular mission. A mission (*M*) consists of several tasks which are constrained by several properties such as start time, deadline, task dependency, and resource requirements. Table 4.2 shows a mission which consists of seven tasks namely τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, and τ_7.

Algorithm bool **AdmissionControl (Node *i*, Task τ)**
    if Node (*i*) has sufficient resource *R* to execute τ **then**
        Allocate *R* in Node (i) for τ
        return *true*
    else
        return *false*
    end if

Algorithm MissionAccomplishment (Mission *M*, Node *i*)
    μ: average RTT (*Round-Trip Time*) between two nodes
    while (*M* <> NULL) **do**
        D = DAG (Directed Acyclic Graph) based on *M* and its task dependency
        T = Tasks belonging to *D* that don't have any dependency
        while (*T*<>NULL) **do**
            τ = pop (*T*)
            if (currentTime (*i*) < startTime (τ) - μ) **then**
                $T = T \cup \{\tau\}$
                continue
            end if
            if (*AdmissionControl* (*i*, τ) = true) **then**
                Node *HashNode* = HashFunction (τ)
                C = CostFunction (*i*, τ)
                Send *C* to *HashNode*
            end if
        end while
        $M = M \backslash T$
    end while

Algorithm NodeSelectionForTask (Node *HashNode*, Task τ)
    HashNode receives all cost values (*C*) from all eligible nodes for *t*
    HashNode selects the node (*i*) with lowest cost value
    HashNode informs the decision to all eligible nodes
    All nodes except node (*i*) release *R* allocated before

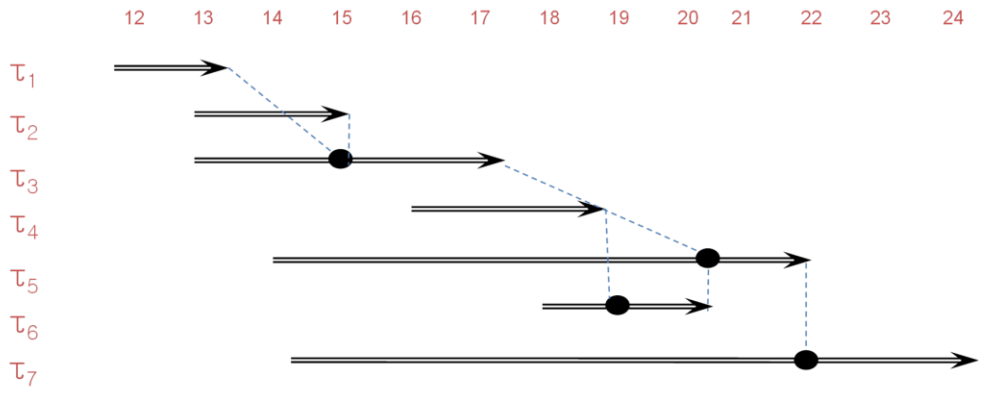
According to *MissionAccomplishment Algorithm*, at first, a Directed Acyclic Graph (DAG) is generated from the mission. To generate the DAG, it is required to have a time–dependent task graph, which shows not only the start and end times of the tasks but also the inter-dependency of the tasks. Figure 4.2(a) presents the time-dependent task graph of seven tasks present in Table 4.2. Figure 4.2(b) presents the corresponding DAG of these seven tasks.

Table 4.2. A simple mission of seven tasks

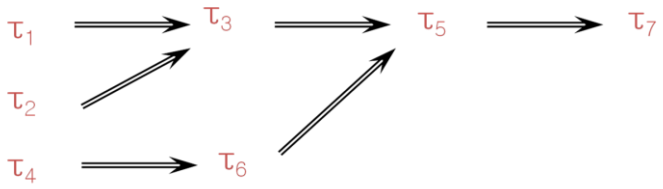
Mission				
Task	**Start**	**End**	**Depends**	**Resources**
τ_1	*12:00*	*13:00*	*X*	R_1, R_1
τ_2	*13:00*	*15:00*	*X*	R_3
τ_3	*13:00*	*17:00*	τ_1, τ_2	R_3
τ_4	*16:00*	*18:00*	*X*	R_4
τ_5	*14:00*	*22:00*	τ_3, τ_6	R_5, R_6
τ_6	*18:00*	*20:00*	τ_4	R_1
τ_7	*14:00*	*24:00*	τ_5	R_2, R_3

From the DAG, a set *T* is generated which consists of the tasks that don't have any dependency on other tasks. For instance, Tasks τ_1, τ_2, and τ_4, don't have any dependency (Figure 4.2(b)) and hence, $T = \{\tau_1, \tau_2, \tau_4\}$. Now, for each task ($\tau$) in *T*, each node (*i*) checks τ's start time. If (*i*'s current time + average round trip time ≈ τ's start time), then node (*i*) executes the *AdmissionControl Algorithm* for τ. As soon as node (*i*) passes through the *AdmissionControl Algorithm*, node (*i*) starts bidding for τ.

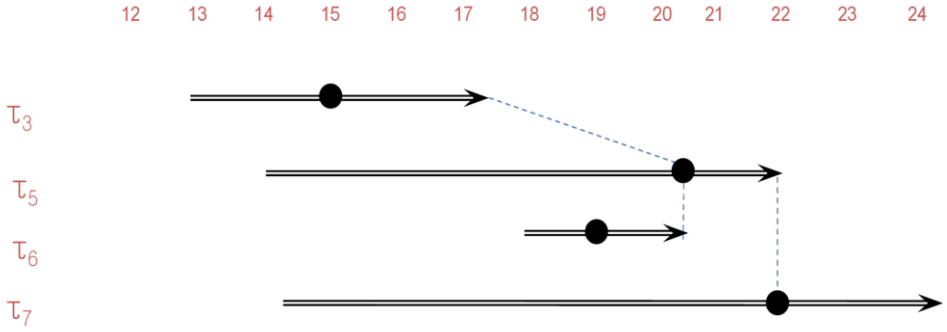
To accomplish that, node (*i*) first finds the *arbiter node* (*HashNode*) based on a predefined hash function. Then node (*i*) calculates its bid value (*C*) by executing the cost function. Bid value (*C*) consists of three parts: *Distance Cost* (C_d), *Bandwidth Cost* (C_b), and *Computation Cost* (C_u). Cost function calculates each part in the following ways:



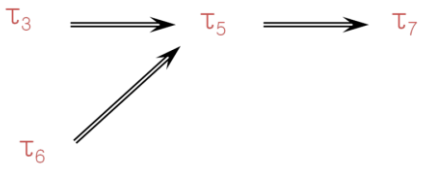
(a) Time-Dependent Task Graph of the seven tasks presented in Table 4.2



(b) Directed Acyclic Graph (DAG) of the tasks presented in Table 4.2



(c) Time-Dependent Task Graph of the remaining four tasks presented in Table 4.2



(d) DAG of the remaining four tasks presented in Table 4.2

(e) DAG of the remaining two tasks presented in Table 4.2

Figure 4.2. Example of how MissionAccomplishment algorithm works

$$C_d = Distance\ between\ source\ and\ destination\ node\ \times\ Weight\ for\ Communication$$

$$C_b = Node's\ Bandwidth\ \times\ Weight\ for\ Bandwidth$$

$$C_u = Node's\ Computational\ Value \times\ Weight\ for\ Computation$$

The weight values for C_d, C_b, and C_u depend on task type. For instance, a purely computational task has the highest weight value (1.0) for C_u whereas the weight values C_d and C_b are 0.0. A bandwidth-hungry task, which also requires communication among nodes, can have a weight value of 0.9 for C_b, a weight value of 0.5 for C_d, and a weight value of 0.0 for C_u. All these weight values are predefined. C_d, C_b, and C_u have different measurement units. To make these values unit-less, C_d, C_b, and C_u should be normalized in the following ways:

$$normalized\ (C_d) = \frac{Distance\ between\ src\ and\ dest\ node}{Maximum\ distance\ between\ two\ nodes} \times Communication\ Weight$$

$$normalized\ (C_b) = \frac{Node'sBandwidth}{Node's\ maximum\ Bandwidth} \times Bandwidth\ Weight$$

$$normalized\ (C_u) = \frac{Node's\ Computational\ Value}{Node's\ maximum\ Computational\ Value} \times Computation\ Weight$$

$$C = normalized\ (C_d) + normalized\ (C_b) + normalized\ (C_u)$$

All other eligible nodes calculate their bid values in the same way and send it to the *HashNode*. The *HashNode* waits for a threshold period to get bid values from all the eligible nodes. As soon as *HashNode* gets all the bid values, it selects the *highest bidder* (*HB*) and sends back the result to all the eligible nodes. The *HashNode* executes the third algorithm (*NodeSelectionForTask Algorithm*) for this purpose. When *HB* gets the response back from the *HashNode*, it starts executing the task. The other eligible nodes release the resources they have allocated earlier, so that they can use the resources for other tasks, if required.

4.5. Example

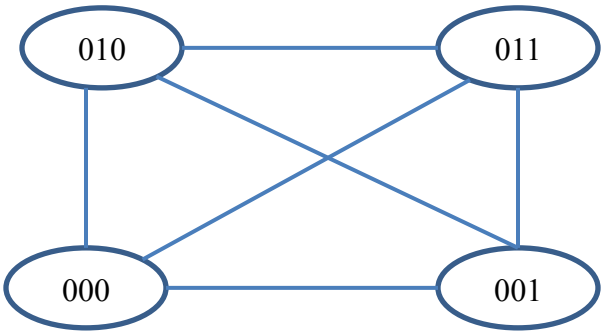


Figure 4.3. Four nodes to accomplish the mission presented in Table 4.2

Suppose, four nodes are assigned to accomplish the mission presented in Table 4.2 and the ids of these four nodes are: namely *000*, *001*, *010*, and *011*. Per our assumption, these nodes form a local full-mesh topology which is shown in Figure 4.3. Figure 4.2 (b) presents the DAG generated from the seven tasks given in Table 4.2. Here, we see that, at the very beginning, only three tasks (τ_1, τ_2, and τ_4) are independent. So, these tasks must be allocated first. Suppose, node *000* and *100* are eligible, i.e., have sufficient resources to perform task τ_1. Both of these nodes will choose, at the start time of task τ_1, the *HashNode* (by using predefined hash function) for task τ_1. Suppose, *HashNode* = Hash (τ_1) = *011*. Now, both of these nodes (*000* and *100*) calculate their bid values considering τ_1's type (e.g., *computationally intensive* versus *communicationally intensive*) and then send the bid value to the *HashNode (011)*. After receiving the bid values from both of these nodes (*000* and *100*), the *HashNode* chooses the *highest bidder*, *HB*. Let, *HB = 000*. Now the *HashNode* informs its decision to both *000* and *100*. As soon as *HB* gets back the response from the *HashNode*, it starts executing the task τ_1. The other node (*100*) releases all the resources it allocated for task τ_1 earlier so that it can use these resources for other

tasks, if needed. In the same way, two other independent tasks (τ_2 *and* τ_4) are assigned to two different nodes through dynamically chosen *HashNode*.

After tasks τ_1, τ_2, *and* τ_4 were successfully allocated to the most eligible nodes, these three tasks are excluded from the Mission (*M*). The new time-dependent task list of *M* is shown in Figure 4.2(c). The corresponding DAG is shown in Figure 4.2(d). This new DAG consists of four tasks (τ_3, τ_5, τ_6, and τ_7) and only two of them are independent (τ_3 *and* τ_6). Now, by exploiting the same approach, these two tasks are assigned through dynamically chosen *HashNode*. Now the newer DAG consists of two tasks (τ_5 *and* τ_7) which is shown in Figure 4.2(e). The same approach is followed for the independent task (τ_5). Finally, only one task is left (τ_7) and is assigned through the *HashNode*. This way, all the seven tasks of the mission have been assigned to the best nodes and the mission is accomplished.

CHAPTER 5: IMPLEMENTATION DETAILS

From the implementation point of view, both centralized and decentralized approaches consist of three modules:

1. Tcl file
2. C file
3. Script file

I have used ns-allinone-2.33 for the experiment.

5.1. Centralized Optimal Task Allocation Algorithm (COTAA)

5.1.1. Tcl file

The Tcl file is used here for the following purposes:

- Get the data (e.g., number of nodes, number of attributes, number of tasks, seed value, and mobility scenario for mobile nodes) from the shell script / command prompt.
- Simulate the hierarchical topology
    - One central node: This node is responsible for collecting all information from the mobile nodes via *InfoBroker* nodes.
    - Four *InfoBroker* nodes: These broker nodes are connected to the *central node* via wired connection.
    - Several mobile nodes: There are two types of *mobile nodes,* namely publisher nodes and subscriber nodes. Each *mobile node* is connected to the corresponding *InfoBroker* wirelessly.

- ***Mobility model***
    - I have used *random way point mobility model* for the mobile nodes. For generating scenarios for *random way point mobility model*, I have used the ns-2 library available at: /ns-allinone-2.33/ns-2.33/indep-utils/cmu-scen-gen/setdest.

For other parameters used in Tcl file, I have utilized the online tutorial [23].

5.1.2. C files

I have designed new communication protocols to provide complex interactions among nodes. Tcl files are not sufficient for this purpose and hence, I have written several C files. As mentioned earlier, there are three types of nodes for COTAA such as the mobile nodes (e.g., publisher and subscriber), the *InfoBroker nodes* and the *Central Controller node*. When a node (let, a mobile node) communicates with other node (e.g., *InfoBroker node*), the communication occurs between corresponding agents. All the C files described below are used by the nodes through their corresponding agents.

- ***Boeing-client.cc***: Each mobile node agent uses the functionality provided by this C file. A mobile node periodically invokes *beacon_callback()* function to generate beacon message informing its *InfoBroker* that it is alive. The *recv()* function is used to receive message (e.g., task assignment, beacon message, etc.) from the *InfoBroker*.
- ***Boeing-gateway.cc*:** Each *InfoBroker* uses the functionality provided by this C file. The *beacon_callback()* function is used to inform the mobile nodes which *InfoBroker* the mobile nodes belong to. InfoBroker invokes *assign()* function to assign a particular task (via packet format) to the specific destination node. The *recv()* function is used for two purposes:

- *InfoBroker* receives a packet from the *central node*. This packet contains the task assignment information. As soon as the *InfoBroker* receives this packet, it sends the packet to the corresponding mobile nodes.
- *InfoBroker* receives a packet from the mobile nodes. This packet contains the resource information of the mobile nodes. The *InfoBroker* accumulates all such packets and then forwards the accumulative result to the *central node*.

When the *InfoBroker* doesn't receive any beacon message from a mobile node for a threshold period, it sends the updated information to the *controller node*.

- ***Boeing-control.cc***: The *central node* uses the functionality provided by this C file. The *controller node* maintains a dynamic table which is increased or decreased based on the number of nodes currently participating in the mission. It invokes *recv()* function to collect nodes' available resource information via *InfoBroker*. It invokes *alloc_task()* function to apply the *Hungarian Algorithm* to the dynamic table to find the optimal task allocation. The *reply_assignment()* function is used to send the task assignment list to the corresponding *InfoBroker*. The *repair_callback()* function is used to repair the task assignment list if a node fails or moves away from wireless communication range.
- ***Boeing-agent.cc***: All these node agents have some common characteristics (e.g., the same packet format). *Boeing-agent.cc* is used to provide such common characteristics.
- ***Hungarian.c***: This file is used by the *controller* node to implement the *Hungarian Algorithm* for OTA.

5.1.3. Script files

There are two shell script files for COTAA, namely *run_static.sh* and *run_mobile.sh*. The *run_static.sh* script is used to run the Tcl file for the static scenario, while *run_mobile.sh* is used for the same purpose, but for the mobile scenario. The code for these two script files is given in the Appendix Section.

5.2. Decentralized Optimal Task Allocation Algorithm (DOTAA)

5.2.1. Tcl file

The Tcl file is used here for the following purposes:

- Get the data (e.g., number of nodes, number of attributes, number of tasks, seed value, and mobility scenario for mobile nodes) from the shell script / command prompt.
- Simulate the hierarchical topology: As mentioned earlier, DOTAA topology is different than COTAA topology which is portrayed in the Tcl file. In DOTAA, several mobile nodes form a group which is directed by a *group leader*. The predefined *group leader* is positioned in the mission area randomly. The other nodes of the same group are placed strategically in such a way that these nodes will be close enough (single-hop away) from the *leader node*. Note that the concepts of the *leader node* and the *arbiter node* are different. While the *leader node* is responsible for the group mobility, the *arbiter node* is used for the task allocation. However, a *leader node* can also be an *arbiter node*.
- ***Mobility model***: All mobile nodes follow their leader for mobility purposes. However, the current version of ns-2 doesn't support such kind of group mobility. Hence, I have prepared a library file to generate such mobility using Visual C#.

5.2.2. C files

- ***Boeing-client.cc***: This file is used to represent all the mobile nodes. The function *genPkt()* is used to generate packets to send to other mobile nodes. The *recv()* function is used to receive two types of packets from the other mobile nodes: *bid packet* and *task packet*. The *bid packet* contains the bidding value for a task by a node and is received only by the *arbiter node*. The *task packet* is received by the destination mobile node. The *command()* function is invoked by each eligible mobile node to send their bid value to the *arbiter node*.

5.2.3. Script files

Like COTAA, there are two shell script files for DOTAA namely *run_static.sh* and *run_mobile.sh*. The *run_static.sh* script is used to run the Tcl file for the static scenario, while *run_mobile.sh* is used for the same purpose, but for the mobile scenario. The code for these two script files is given in the Appendix Section.

CHAPTER 6: EXPERIMENTS AND RESULTS

6.1. Centralized Optimal Task Allocation Algorithm (COTAA)

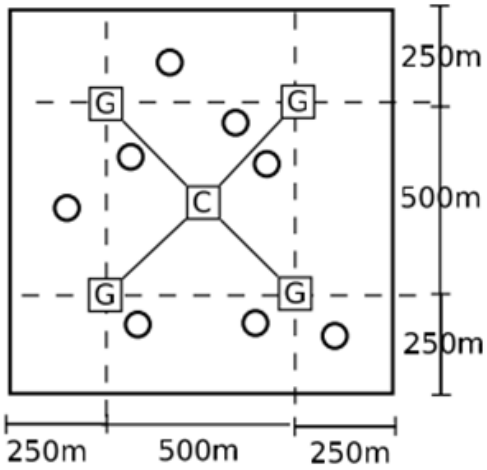


Figure 6.1. Evaluation Scenario for COTAA

6.1.1. Evaluation Scenario and Parameters used for COTAA

I evaluate the performance of COTAA via simulations using ns-2 simulator. The simulation runs for the first-responder rescue mission on 1km x 1km area as depicted in Figure 6.1. There are four static *InfoBrokers* (i.e., rescue vans), depicted by "G" box, distributed evenly in the area. Each node (i.e., rescue agent) communicates to one of the *InfoBrokers* wirelessly. Each *InfoBroker* is connected to the central unit, *CU* (i.e., rescue headquarter), depicted by "C" box in the picture, via direct interface (i.e., dedicated radio or directional antenna). Unless otherwise specified, the default parameters used in the simulation are presented in Table 6.1. The tasks in the scenario consist of either publisher tasks (i.e., publishing sensor information) or subscriber tasks (i.e., receiving sensor information). Each task (j) has its own utility (U_{ij}), which is a linear function of the resource parameter of node (i). In the simulation, the resource status of each node ($P_{ik},\ \forall k \in P$) and the corresponding weights ($W_{jk},\ \forall k \in P$) are generated uniformly and randomly.

I evaluate the performance of the proposed task assignment protocol in terms of *end-to-end delay* and *bandwidth overhead*. The *end-to-end delay* is the delay between the time when the task assignment request is issued at *CU* and the time that all task assignment messages arrive at the nodes. The *bandwidth overhead* is the bandwidth consumption incurred from the node status messages and task assignment messages.

Table 6.1. Simulation Parameters for COTAA

Parameters	***Values***
Number of Nodes	64-448
Number of Resource Parameters	32
Number of Tasks	1-32
Wireless Transmission Range	250 m
Mobility Model	Random Waypoint
Speed	1-20 m/s
Advertisement Period	5 s
Simulation Period	500 s
Number of Runs per instance	5

6.1.2. Static Nodes

I first present the result in the scenario where nodes do not move. I conduct the experiment under this scenario in order to measure the scalability of the protocol in terms of number of nodes and tasks without the effect of mobility. From Figures 6.2 and 6.3, the following conclusions can be drawn about the *end-to-end delay*. First, although the theoretical computation overhead in the *task assignment algorithm* is $O(n^2m)$, where n is the number of nodes and m is the number of

tasks, the computation delay is much smaller than the communication delay in a wireless network setting. This fact is confirmed in Figure 6.2, as the communication delay grows linearly as the number of tasks increases. On the other hand, the computation delay grows much more slowly as the number of nodes in the system grows, as seen in Figure 6.3. From this result, we can conclude that the delay bottleneck is the number of tasks to be assigned, as it affects communication delay. The same conclusion can be drawn for the bandwidth overhead, which is presented in Figure 6.4. The bandwidth consumption grows linearly with the number of tasks and the number of nodes in the system.

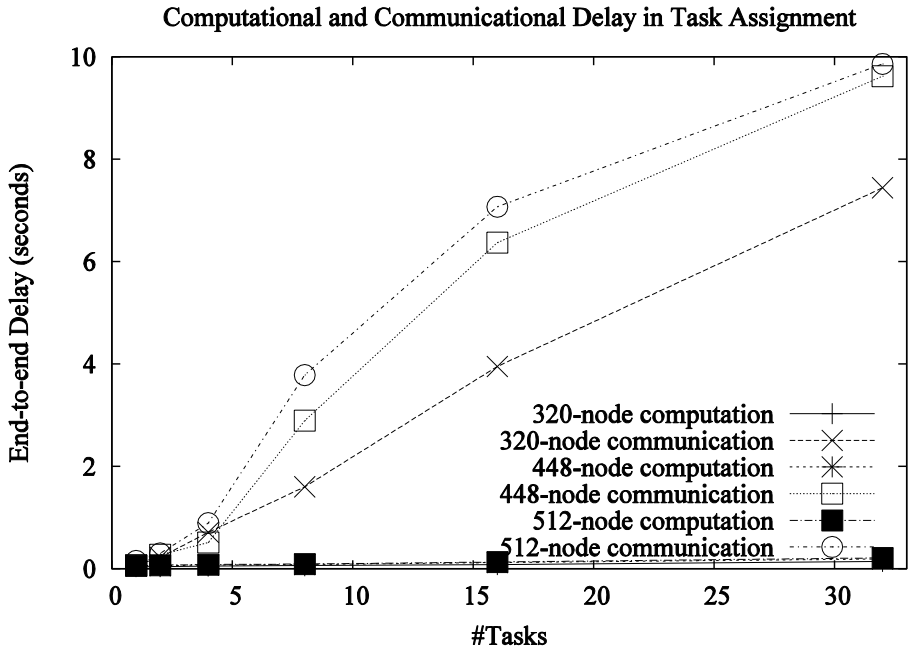


Figure 6.2. COTAA (Static Nodes): End-to-end Delay vs. Tasks

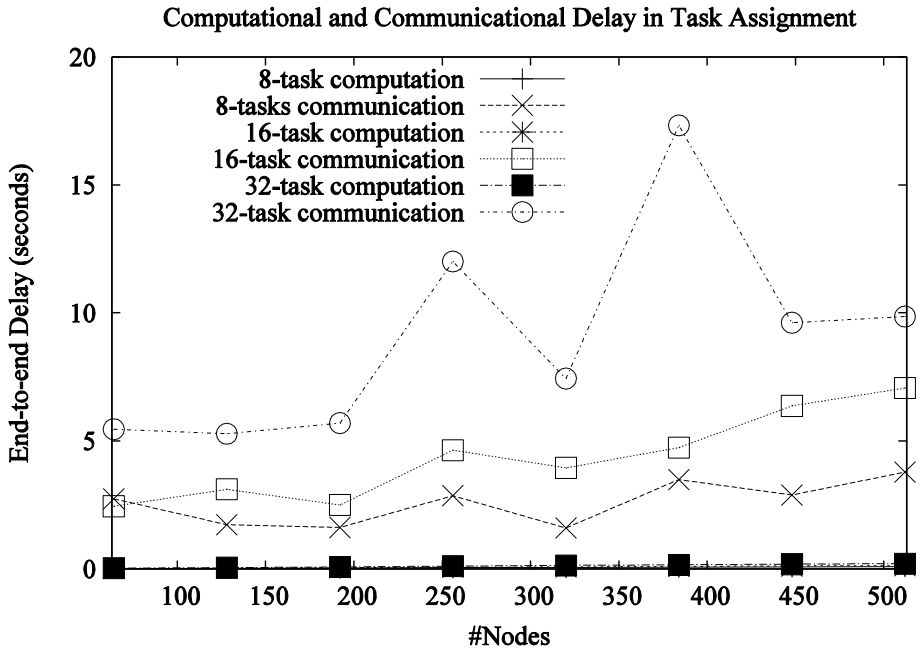


Figure 6.3. COTAA (Static Nodes): End-to-end Delay vs. Nodes

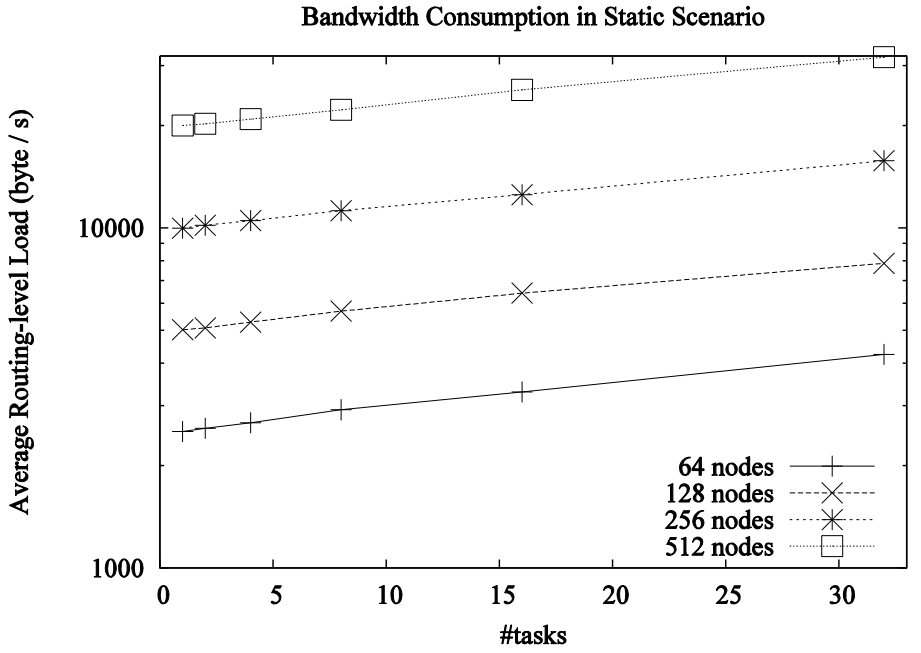


Figure 6.4. COTAA (Static Nodes): System Load vs. Tasks

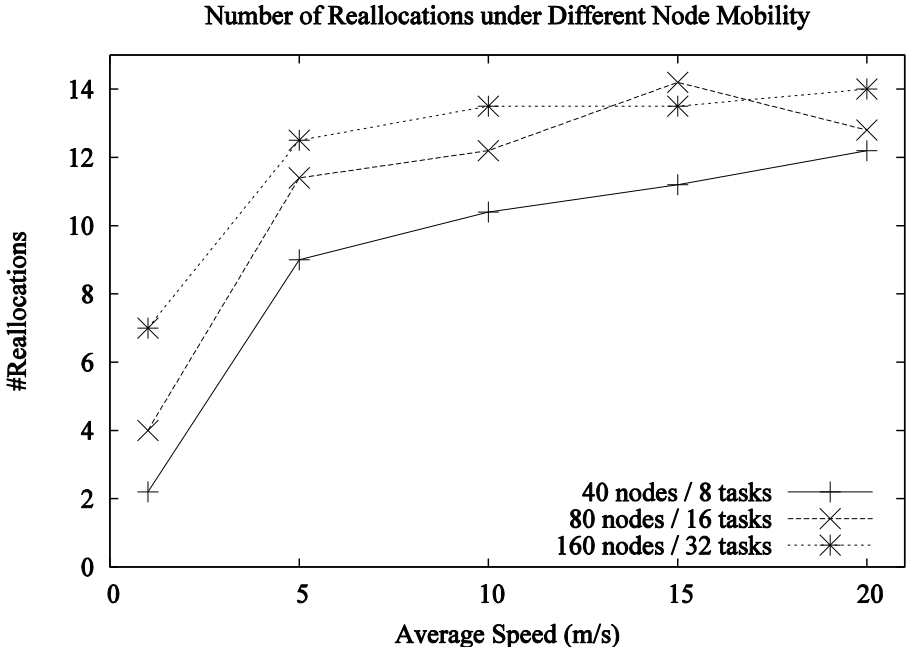


Figure 6.5. COTAA (Mobile Nodes): # of Reallocation vs. Average Speed

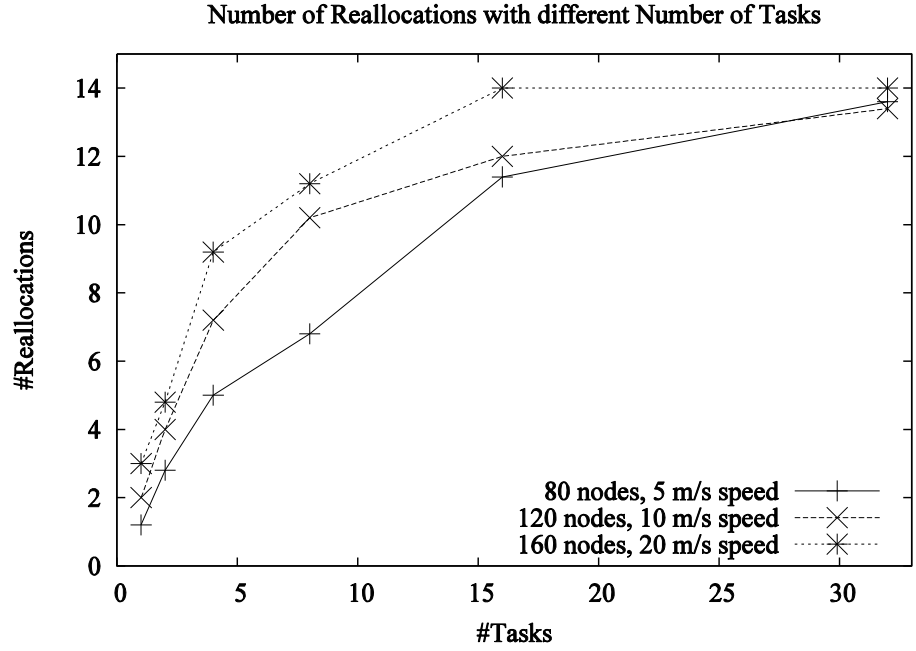


Figure 6.6. COTAA (Mobile Nodes): # of Reallocation vs. # of Tasks

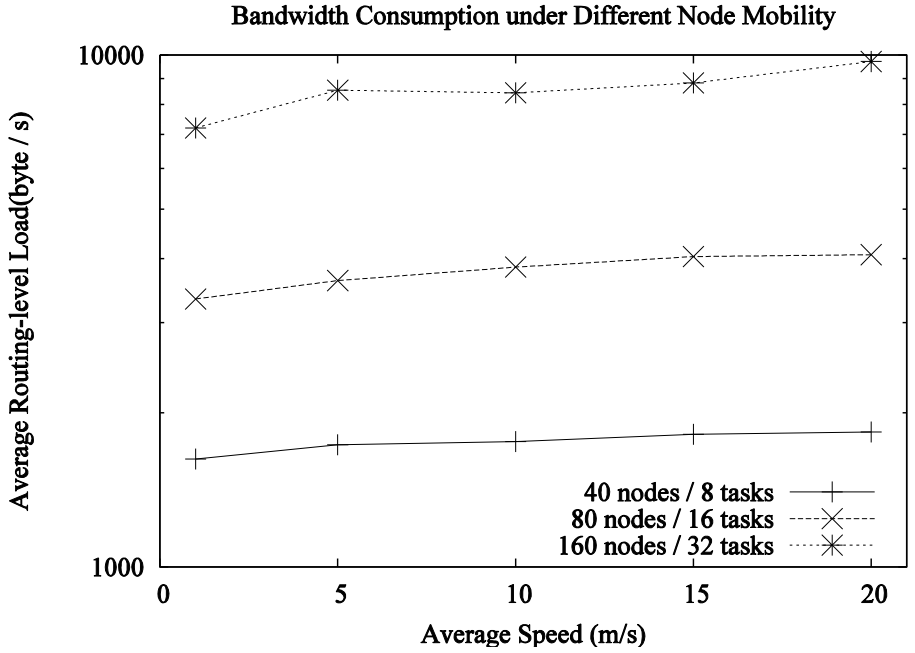


Figure 6.7. COTAA (Mobile Nodes): System Load vs. Average Speed

6.1.3. Mobile Nodes

This section presents the performance of the task allocation protocol depicted in Figure 6.1, but with nodes moving around the area using *random way point mobility model*. Albeit it might not be possible to have an ideal mobility model which can deal with all the possible scenarios of wireless networks, I believe the *random way point mobility model* is a reasonable assumption as in MCE, nodes are distributed in a heterogeneous manner and their movements are relatively random. As described earlier, *CU* will check for lost tasks due to node mobility and perform the re-allocation periodically. I set that time interval to 30 seconds in the experiment. Decreasing this interval surely would decrease the outage time resulting from mobility. However, it would also incur additional overhead due to having more message communication. Figure 6.5 presents the performance of the protocol under mobile node scenario. The performance is measured in terms

of the *number of reallocations* through the entire simulation period (500 seconds) and the *system bandwidth consumption*. The controlled parameters are the average node speed, the number of tasks, and the number of nodes in the system.

According to the result in Figures 6.5 and 6.6, we can conclude that with an increase of the node mobility, the number of tasks, or the number of nodes, and the number of re-allocations also increases. However, since the re-allocation algorithm is executed only every 30-second interval and the simulation period is 500 seconds, the maximum number of re-allocations in the experiment is $\left\lfloor \frac{500}{30} \right\rfloor - 1 \approx 15$ times, which is consistent with the simulation results.

The average system load under different mobility level is presented in Figure 6.7. In general, the system bandwidth consumption of the system grows proportionally to the system size. The system bandwidth consumption also grows as mobility level increases, but with a lower rate than the system size. The explanation is that the system load mainly consists of per-node beaconing messages rather than re-allocation messages, which results from the node mobility and subsequent task losses.

6.2. Decentralized Optimal Task Allocation Algorithm (DOTAA)

6.2.1. Evaluation Scenario and Parameters used for DOTAA

DOTAA is applicable for various mission critical scenarios. In this thesis, I present two representative scenarios (e.g., a battlefield scenario and a group of UAV scenario) where DOTAA can be very effective.

Suppose, a critical mission in the battlefield (Figure 6.8) needs to be executed within a specific amount of time. The commander-in-chief, at the very beginning, informs the details of

the mission (e.g., how many tasks are involved, what are the requirements of each task, what is the deadline, etc.) to each soldier. However, since a battlefield is a very dynamic environment, the optimal task allocation might not be possible at the very beginning. Rather each soldier needs to dynamically decide which task he or she will perform considering the situation. Each soldier executes DOTAA in their mobile nodes and gets the decision very quickly from the dynamically chosen *arbiter node* and then starts performing the task (assigned by the *arbiter node*) and updates the resources their nodes consume.

Figure 6.8. A Battlefield Scenario [collected from web]

Figure 6.9. Group of UAV [collected from web]

DOTAA is also applicable for other MCEs. Figure 6.9 presents the operation by a group of UAVs, where each UAV (Unmanned Aerial Vehicle) follows the speed and direction of the predefined leader UAV for speed and direction. Each UAV is dynamically assigned some specific tasks to achieve a particular mission. Each UAV runs DOTAA and bids for a task if it is eligible for that task. By calculating the hash function, each eligible UAV finds the *arbiter node (NH)* and then sends their bids to *NH*. After receiving the bidding values, *NH* finds the most eligible UAV (*A*) for that task. As soon as *A* receives confirmation from *NH*, *A* starts executing the task.

Table 6.2. Simulation Parameters for DOTAA

Parameters	***Values***
Number of Nodes	4-100
Number of nodes per group	4
Number of Tasks per group	1-30
Wireless Transmission Range	250 m
Mobility Model	Group-Based Random Waypoint
Speed	1-50 m/s
Simulation Area	1 km x 1km to 10 km x 10 km

6.2.2. Static Nodes

Similar to COTAA, I first evaluate DOTAA for the scenario where nodes do not move. This experiment has two major goals.

1. To evaluate the scalability of DOTAA in terms of number of nodes, number of tasks, and the geographical area.
2. I envision a very specific kind of MCE where nodes are strategically placed at the very beginning of the mission and hence, nodes will rarely move during the entire mission.

For this experiment, the leader nodes are first placed randomly in mission's geographical region and the corresponding eligible nodes are placed accordingly to ensure that these nodes are single-hop away from the leader node.

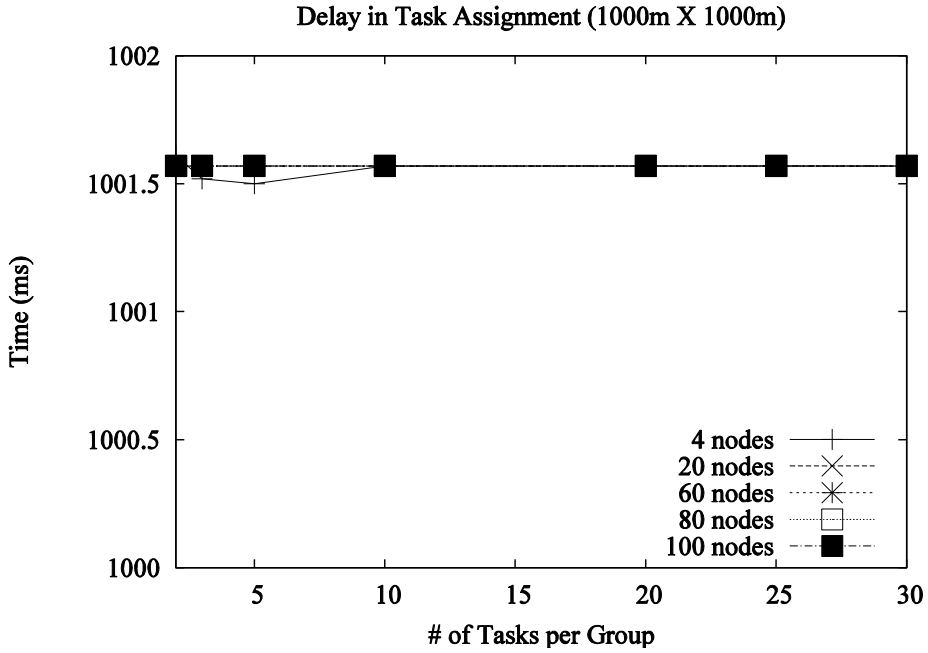


Figure 6.10. DOTAA (Static Nodes): Task Assignment Time vs. # of Tasks/Group (1km x 1km)

Figure 6.10 presents the impact of the number of tasks per group to the task allocation time for the static scenario. I consider 1 km x 1 km geographic area for this experiment. The task allocation time has two parts: *Computation Time* and *Communication Time*.

Figure 6.10 shows that the number of tasks has little impact on the task allocation time. However, we saw earlier that, in case of COTAA, the number of tasks has great impact, especially, on the communication time since multi-hop communication is required between *mobile nodes* and the *controller node*. For DOTAA, the *arbiter node* and the *mobile nodes* are always single-hop away. The task allocation time of DOTAA lies between 1001.5 ms to 1002.0

ms which is significantly less than COTAA. This result indicates that DOTAA is more scalable than COTAA.

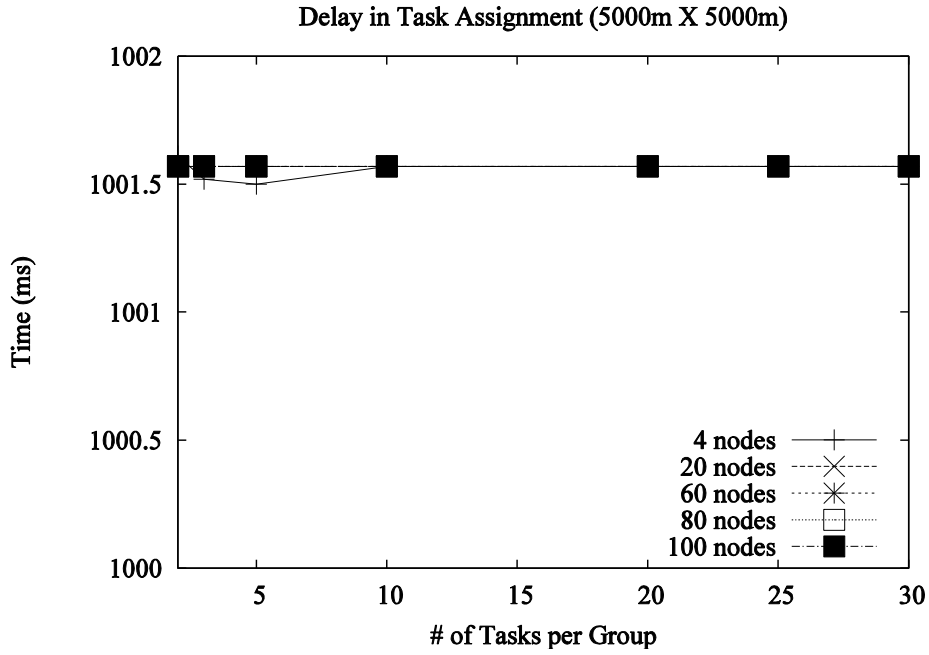


Figure 6.11. DOTAA (Static Nodes): Task Assignment Time vs. # of Tasks/Group (5km x 5km)

Figures 6.11 and 6.12 present the impact of geographical area to task allocation time for static scenario. Here, we see that, for other geographical areas (e.g., 5km x 5km and 10 km x 10 km), the task allocation time remains similar. The reason is that, even when the area is increasing, each node in a particular group maintains the same full mesh topology and no group interferes with other group for communication purposes. Hence, the task allocation time does not vary much.

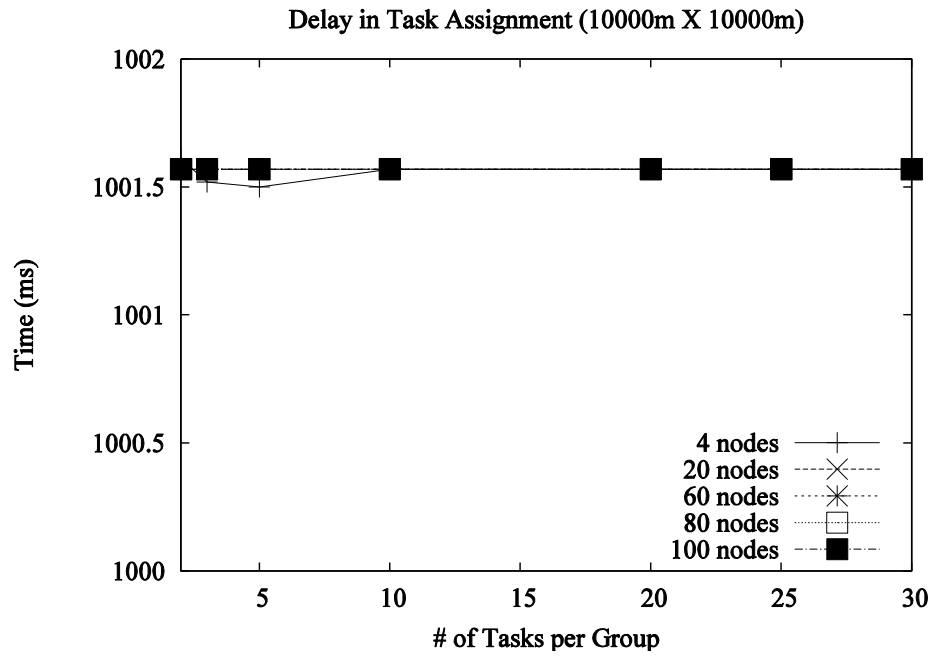


Figure 6.12. DOTAA (Static Nodes): Task Assignment Time vs. # of Tasks/Group (10km x 10km)

Figures 6.13, 6.14, and 6.15 present the average routing-level system load per node (byte/sec) for static scenario. The geographical areas considered here are: 2 km x 2 km, 5 km x 5 km, and 10 km x 10 km. The result largely differs from the result of COTAA. In case of COTAA, the system bandwidth consumption of the system grows proportionally to the system size. On the contrary, for DOTAA, the average bandwidth consumption doesn't depend on the system size. The reason is that, in such full mesh topology, each node can send the packet to the destination node within one hop and hence packet loss rarely happens. As I mentioned earlier, nodes do not fail during the mission in case of DOTAA. So the average bandwidth consumption per node remains same even for higher system size.

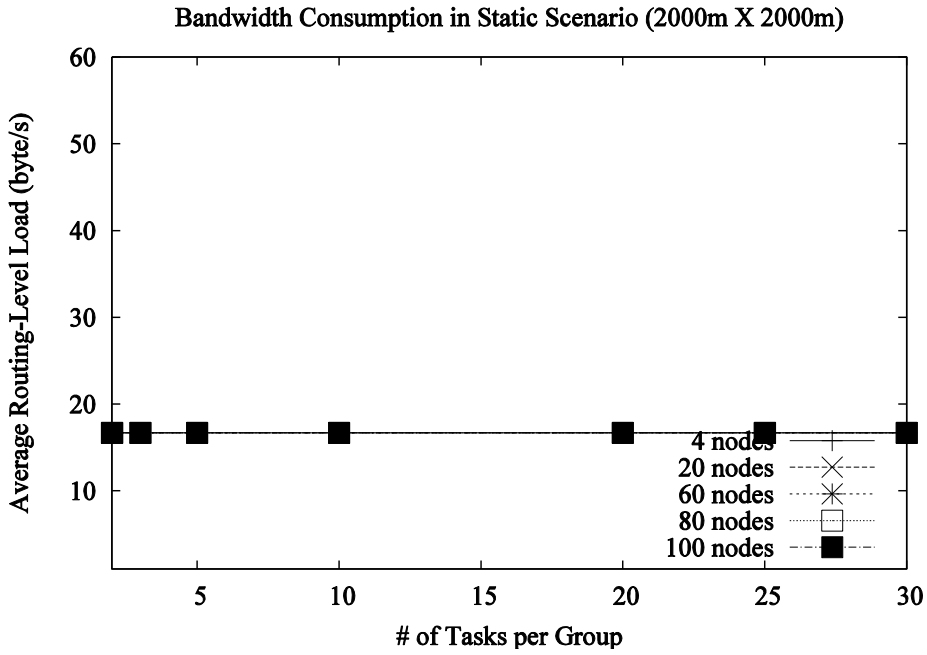


Figure 6.13. DOTAA (Static Nodes): System Load vs. # of Tasks/Group (2km x 2km)

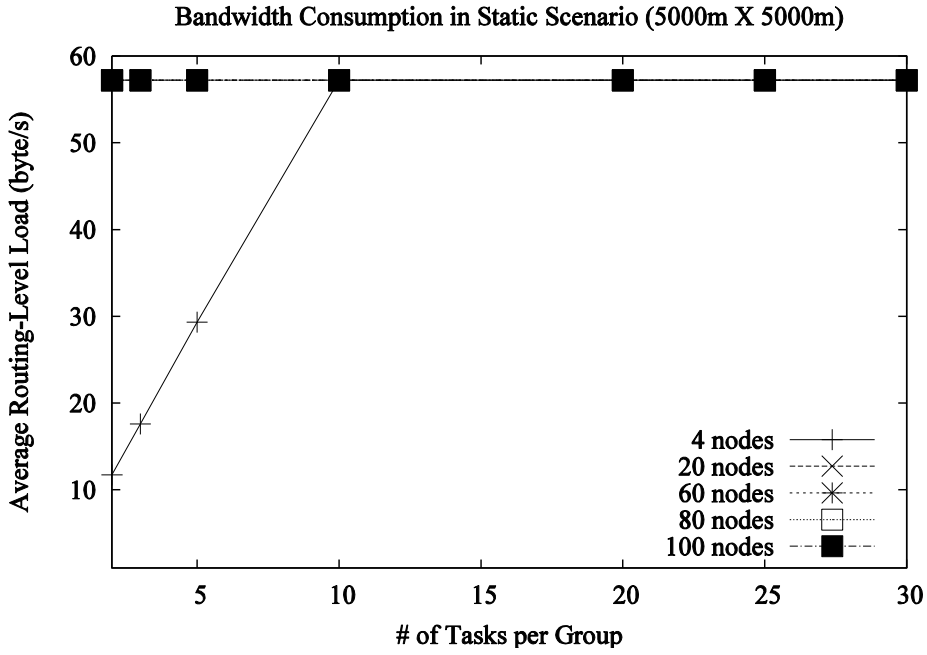


Figure 6.14. DOTAA (Static Nodes): System Load vs. # of Tasks/Group (5km x 5km)

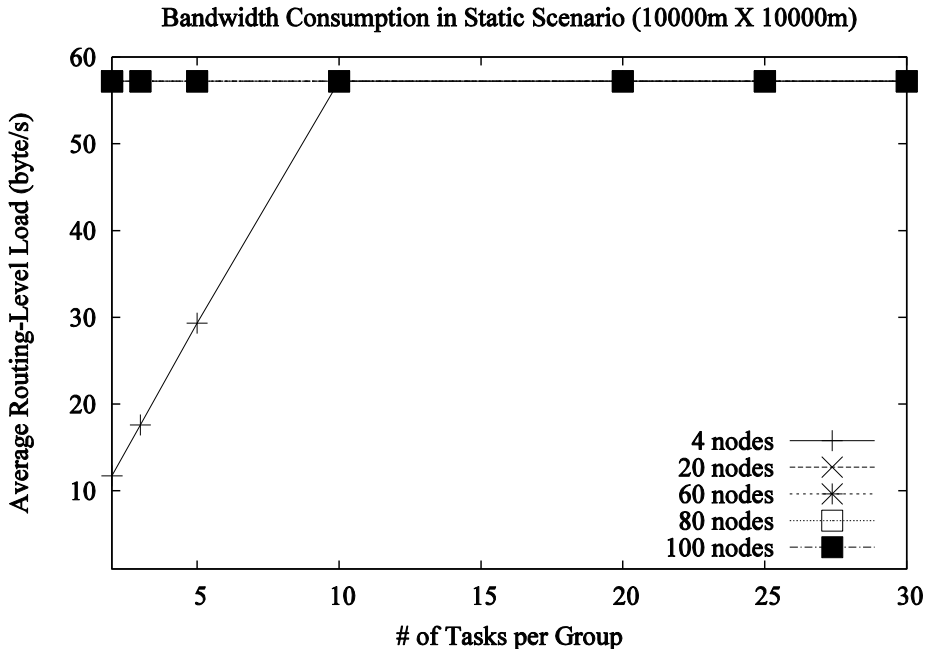


Figure 6.15. DOTAA (Static Nodes): System Load vs. # of Tasks/Group (10km x 10km)

6.2.3. Mobile Nodes

In case of the battlefield scenarios, firefighting situations, post-disaster recovery, etc., people normally move in groups. They normally follow their leader or they move together by communicating with each other. In short, the relative distance among nodes becomes nearly constant even for very high speed. Figure 6.16 presents such a group-based mobility scenario.

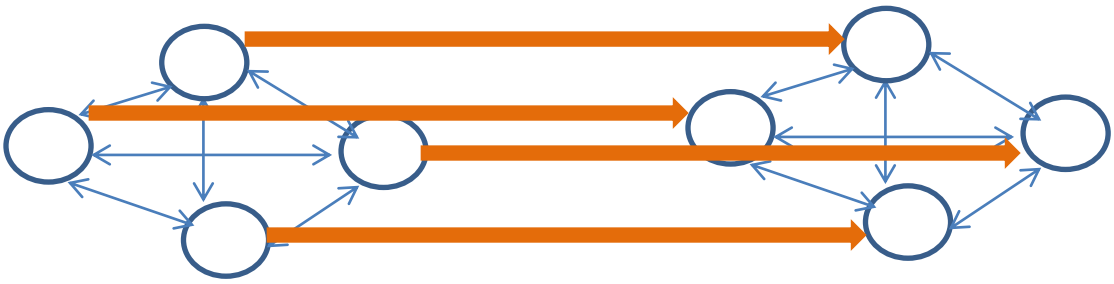


Figure 6.16. Group wise Movement

Figures 6.17, 6.18, and 6.19 present the impact of the average node speed to the task allocation time for the mobile scenario. Here, I consider several geographic areas such as 1 km x 1 km, 2 km x 2 km, and 10 km x 10 km. The results show that the task allocation time varies between 1001.5 ms to 1002 ms. Albeit the node speed is very high, since the relative distance among nodes remains almost constant, the packet delivery time is almost same as in the static scenario.

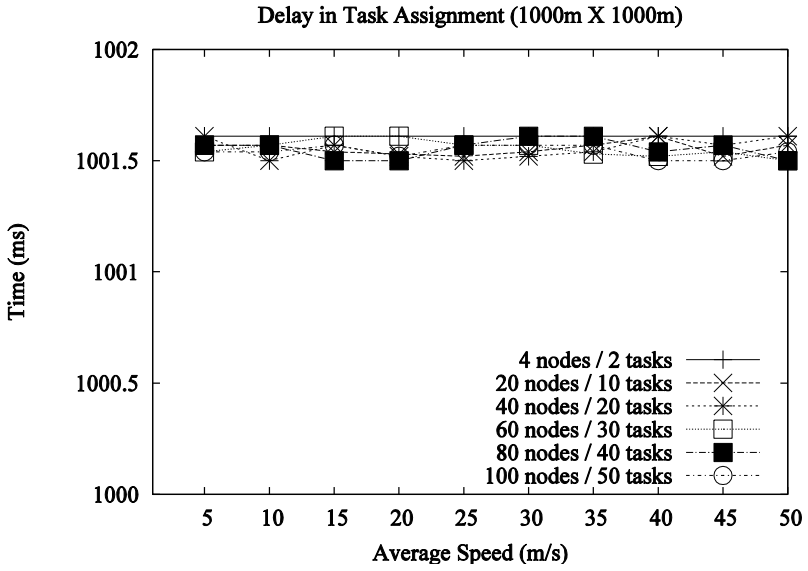


Figure 6.17. DOTAA (Mobile Nodes): Average speed vs. task allocation time (1km x 1km)

Figures 6.20, 6.21, and 6.22 show the average routing-level system load per node (byte/sec) for the mobile scenario. The geographical areas considered here are: 1 km x 1km, 3 km x 3 km, and 5 km x 5 km and the average speeds of the nodes range from 5 m/s to 50 m/s. Here, 5 m/s can be considered as the pedestrian speed while 50 m/s represents the UAV speed. In these Figures (6.20 to 6.22), we see that, the system load is very low when the number of nodes and tasks is small (4 nodes/2 tasks). However, even when the speed is very high, the average system load is no more than 60 bytes/s. Even for higher speeds, the relative distance among nodes in a group is

nearly same. Hence, the local communication among nodes is not interrupted due to the speed and most of the packets can be reached to the destination node without any problem.

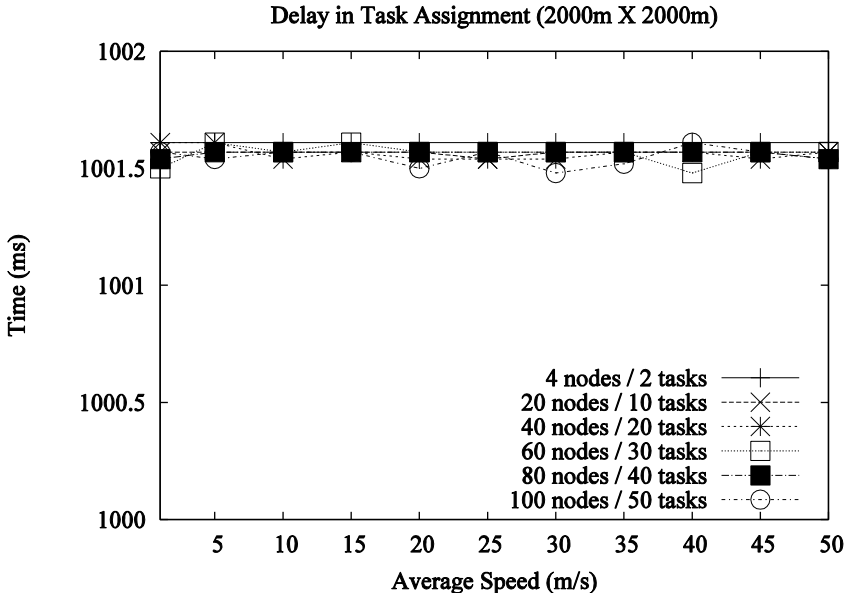


Figure 6.18. DOTAA (Mobile Nodes): Average speed vs. task allocation time (2km x 2km)

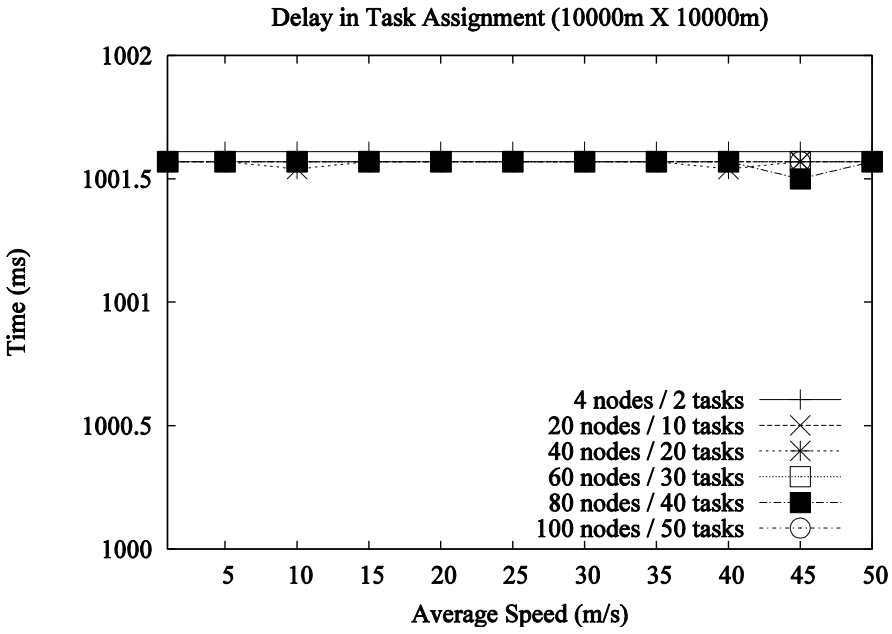


Figure 6.19. DOTAA (Mobile Nodes): Average speed vs. task allocation time (10km x 10km)

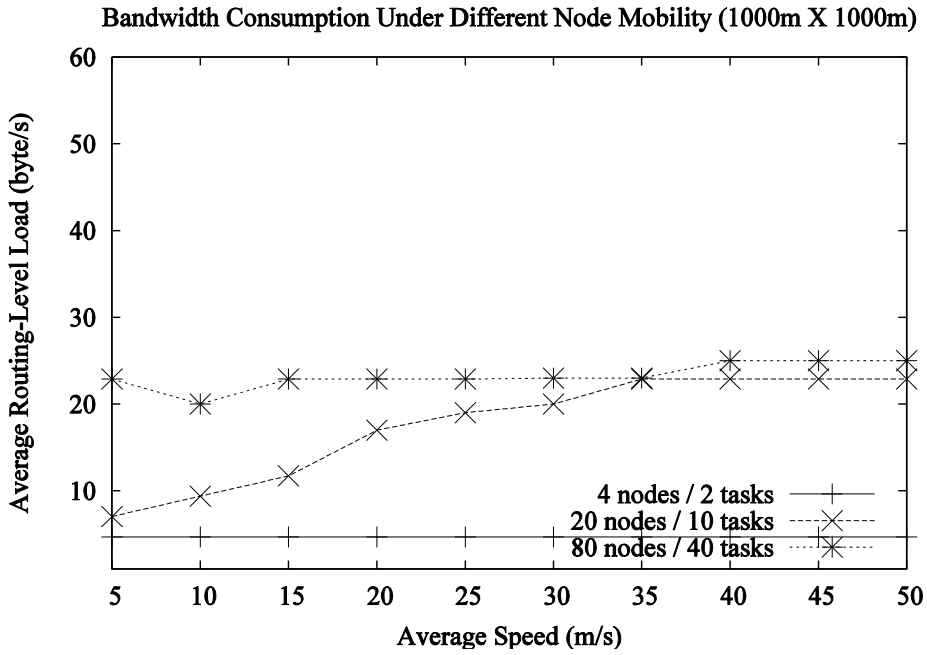


Figure 6.20. DOTAA (Mobile Nodes): Average speed vs. System Load (1km x 1km)

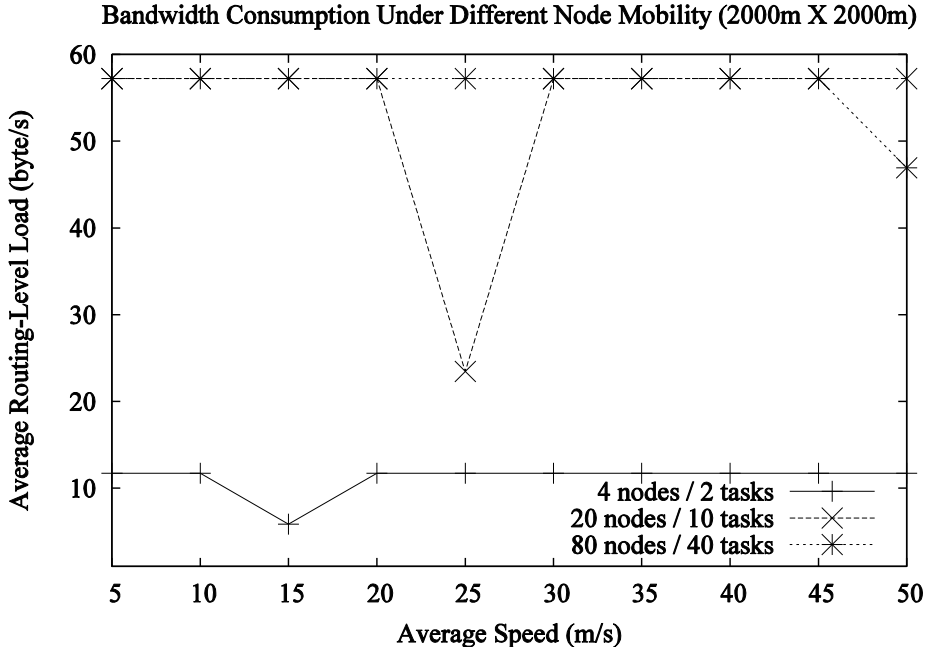


Figure 6.21. DOTAA (Mobile Nodes): Average speed vs. System Load (2km x 2km)

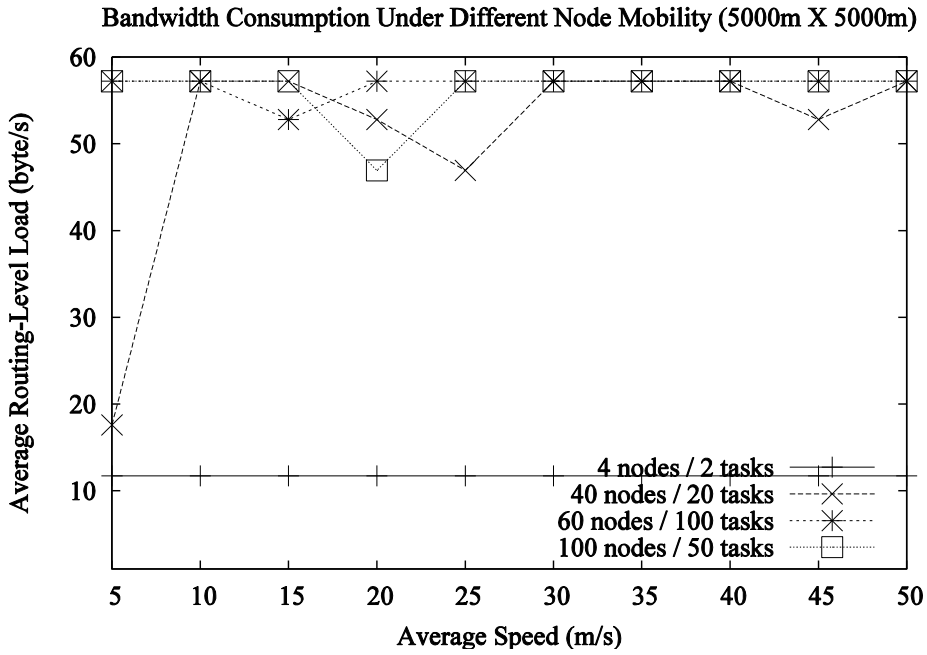


Figure 6.22. DOTAA (Mobile Nodes): Average speed vs. System Load (5km x 5km)

6.3. Discussion

In this thesis, I have described both the centralized (COTAA) and decentralized (DOTAA) approaches. Both of these approaches have some pros and cons.

6.3.1. Pros of COTAA

1. COTAA ensures the global OTA. The central node has the updated information of all the resources of all the nodes. Hence, it is easier for the central node to find the best possible node for a particular task.
2. COTAA is particularly useful for smaller group of nodes.
3. COTAA is better applicable for MCE, where global optimality is more important than local optimality.

6.3.2. Cons of COTAA

1. COTAA converges slower than DOTAA, especially for larger number of nodes. COTAA needs several seconds for performing optimal task allocation.
2. COTAA is vulnerable to the single point of failure. If the controller node fails, the entire system becomes dysfunctional.

6.3.3. Pros of DOTAA

1. DOTAA ensures fast task allocation (around one sec only) as opposed to COTAA.
2. DOTAA is more scalable than COTAA. The task allocation time for DOTAA does not increase much with the increase in the number of nodes and tasks.
3. The bandwidth consumption is less in DOTAA. Here the task allocation is performed in the local group only and hence, nodes don't need to send more messages for the task allocation. However, since the increase in the number of nodes and tasks may increase collisions, the bandwidth consumptions increase with the increase in the nodes and tasks.
4. DOTAA is free from single point of failure.
5. DOTAA is better applicable for MCEs where performance is preferred than global optimality.

6.3.4. Cons of DOTAA

1. DOTAA only ensures local OTA. The *arbiter node* of a group receives all nodes' information within the same group. Hence, there might exist some other nodes in other groups which are better eligible for the particular task.

CHAPTER 7: CURRENT STATE OF ART

Based on task allocation, we can classify the existing research directions in several domains:

- Task allocation in Robotics
- Task allocation in UAV
- Mobile-based DHT
- Sensor Networking
- Miscellaneous (e.g., Grid computing, QoS)

7.1. Task Allocation in Robotics

Researchers are investigating the task allocation pattern of multiple robots (MRTA: Multi Robot Task Allocation), especially focusing on the communication pattern among the robots [5, 6]. In [6], the authors described MRTA problem from the formal point of view and provided a very elaborate MRTA taxonomy. They discussed how task allocation in multi-robot situation can be considered as an optimization problem. They also offered some guidelines for how existing theories especially from operations research and combinatorial optimization area can be exploited to solve MRTA problem. The same authors conducted an extensive comparative study in [5] where they elaborated the pros and cons of all existing solutions for MRTA problem.

Sariel-Talay et al. proposed a cooperative framework to solve multiple traveling robot problem (MTRP) based on dynamic task selection and robust execution [28]. MTRP is nothing but the well-known Multiple traveling salesman problem, in context of robotics. The authors followed an incremental task allocation method to adopt the dynamic environment. The same authors proposed another distributed Multirobot-cooperation framework named DEMiR-CF for

autonomous underwater vehicles (AUV) [29]. They have also some other research in the same direction which can be found in [30, 31].

7.2. Task Allocation in UAVs

In [32], Bertuccelli et al. proposed an algorithm for the task allocation in UAVs. Their algorithm, named the Consensus Based Bundle Algorithm (CBBA), is based on conflict-free assignment algorithm. Through CBBA, they tried to create collision free paths for the UAVs. They simulated the real-time performance of CBBA by integrating it with 3D visualization and interaction software tool.

Choi et al. [33] proposed a consensus-based decentralized algorithm for task allocation in UAVs. A market-based decision strategy is exploited to reach a consensus among the UAVs on the winning bid value. The authors claimed that their proposed algorithm ensures the conflict-free feasible solution. They also showed that consensus-based algorithm converges faster than the existing auction-based task-allocation algorithms.

7.3. Mobile-based DHT

DHT-based protocol is prominent in peer-to-peer (P2P) networking. Mobile Ad-Hoc Network (MANET) and P2P share some common characteristics. There is no central infrastructure in P2P and MANET. The topology changes over time due to churn for P2P and due to node mobility for MANET. By observing these similarities and the prominence of DHT-based protocol for P2P, several researchers have been trying to adapt DHT in MANET. Pucha et al. proposed Ekta, a DHT-based algorithm applicable for a distributed application in MANET [19]. They proposed two design options for Ekta. According to the first option, DHT is overlayed in a MANET. Here multi-hop routing protocol is considered. In the second option, the authors examined the

performance of resource discovery application which exploits the physical layer broadcast as opposed to the network layer broadcast.

Zahn et al. [20] proposed MADPastry which, like Ekta, exploits the concept of DHT to use in MANET. One additional advantage of MADPastry over Ekta is that MADPastry considers physical locality i.e. if two mobiles nodes are in the vicinity in the overlay network, MADPastry ensures that those two nodes are also close to each other in the physical locality. Araújo et al. proposed another DHT-based approach for MANET which they named Cell Hash Routing (CHR). CHR has some fundamental differences from Ekta and MADPastry. CHR addresses the problem of limited available energy. CHR uses position-based routing which exploits the node clusters rather than individual nodes. The authors claimed that CHR is more scalable than any other existing mobile-based DHT since CHR uses localized routing and introduces a new concept of load sharing.

7.4. Sensor Networking

Due to energy and resource constrained nature of the sensor nodes, efficient task allocation is an important research topic in the area of sensor network [11, 12, 13]. Most of the researchers of sensor network mainly concentrated on energy efficiency. The goal of the research conducted by Younis et al. [11] was also focused on the same direction. However, while most researchers concentrated on energy efficiency on sensor node itself, Younis et al. paid attention to the energy efficiency at the gateway. To do that, they proposed a task allocation approach to the gateway which maximizes not only the life of the gateway but also the life of the entire sensor network. Yu et al. proposed an energy-balanced task allocation for sensor network [12]. They mainly offered two approaches in this regard: Integer Linear Programming and 3-phase heuristic

executable in polynomial time. They claimed that their proposed approaches work for both small (<=10 tasks) and large systems (60-100 tasks) where lifetime of the sensor network is improved by 5 times for small systems and 3.5 times for large systems. Zhao et al. also proposed a task assignment algorithm for sensor network which is topology-aware, energy efficient, and viable for resource constraints [13]. They named their approach as TETA (Topology-Aware Energy Efficient Task Assignment). They first proved that TETA is NP-complete and then described their ant-based heuristics to solve it. They evaluated their approach through simulation.

7.5. Miscellaneous

From the research point of view, the task allocation and resource allocation have many overlaps. Both of these approaches try to find the best node of a network (wired/wireless) either for the resource allocation or for task allocation. Hence, solutions for task allocation can be applicable to the resource allocation and vice versa. Considering that, I am describing here some research in the domain of resource allocation. Shu proposed an approach for grid computing to ensure the optimal resource allocation [10]. The author exploited the concept of quantum chromosomes genetic algorithm in this regard and evaluated the proposed approach through extensive simulation. Caramia et al. proposed an economic model for resource allocation in grid computing [35]. Their business model is based on tender/contract-net model which involves the interaction among the nodes occurred in the grid computing. They evaluated their approach and compared the result with traditional round-robin allocation algorithm.

CHAPTER 8: CONCLUSION AND FUTURE DIRECTIONS

Mission Critical Environment (MCE) poses numerous challenges which include, but are not limited to, DIL (Disconnected, Intermittent, Limited) communications among nodes, high error rate and data loss, heterogeneous capabilities and resources of mobile nodes, deadline-driven mission, dynamic nature of topology formation and deformation, and random or group mobility with greatest speed [7]. To address these challenges, mission operators (e.g., commander-in-chief of a military force in a battle-field, rescue leader of a post-disaster recovery mission, etc.) periodically execute task allocation manually which is very slow, error-prone, and cumbersome. To address this, in this thesis I have proposed two novel automated, fast, and efficient algorithm, namely, Centralized Optimal Task Allocation Algorithm (COTAA) and Decentralized Optimal Task Allocation Algorithm (DOTAA).

In case of COTAA, I present an architectural framework and communication protocols to solve the Optimal Task Allocation (OTA) problem in the publish/subscribe-based MCE. I exploit the well-known *Hungarian Algorithm* and *Rectangular Assignment Algorithm* to solve the OTA problem in polynomial time. I show that COTAA achieves the goal of OTA while maintaining efficiency, scalability, and reasonably low processing and communication overhead. However, COTAA is based on some assumptions such as: mobile nodes have to follow the communication pattern of a publish/subscribe-based system, tasks in a mission have no inter-dependency, and the central unit is solely responsible for the task allocation. Also, COTAA is vulnerable to a single point of failure.

To relax the assumptions made by COTAA, I propose another novel automated approach named Decentralized Optimal Task Allocation Algorithm (DOTAA). I have exploited the

concept of application-layer *Distributed Hash Table (DHT)* to perform the task allocation. I have extensively evaluated both COTAA and DOTAA using ns-2.

The major lessons I have learned from this thesis are as follows:

1. Cost function plays a critical role both for COTAA and DOTAA. Hence, it is utmost important to design the cost function in a very efficient manner.
2. Both COTAA and DOTAA have pros and cons. DOTAA is better applicable for MCEs where performance is preferred than optimality. On the contrary, COTAA should be chosen when global optimality is preferred.
3. For distributed approach, it is difficult to integrate both DHT-based approach and multi-hop communication.
4. My solutions can be extended in some other domains. For instance, both COTAA and DOTAA can be exploited to provide temporary communication infrastructure for remote areas/underdeveloped regions.

In future, I plan to extend my current research in the following ways:

1. **Hybrid approach**: Combine COTAA and DOTAA to take advantages of both of these approaches. The main idea is that, each node of the group will move together as DOTAA suggests and there will be a central controller unit as COTAA suggests. Each group leader will have both internal and external knowledge of the MCE and the arbiter node can be chosen via inter-leader communication not only from within group but also from outside of the group. The central node will be used for control purposes only.
2. **Multi-Node, Single-Task**: In some MCEs, more than one node might be required to perform a single task. I will provide the support for such multi-node, single-task operation in both COTAA and DOTAA.

3. **Strategic node placement and movement**: By strategy, here I mean, even before the mission starts, each of the nodes will have the tentative plan where it should move next. Then COTAA or DOTAA can perform the task allocation effectively. In this way, the adaptation for the mission and hence, the system load would be minimal, which is a major issue for COTAA.

BIBLIOGRAPHY

[1] J. Bijsterbosch and A. Volgenant, "Solving the Rectangular assignment problem and applications", URL: www.optimization-online.org/db_file/2008/10/2115.pdf

[2] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling", In In the Proceedings of 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), pp. 213-223, 2005.

[3] A. Belokosztolszki, D. Eyers, P. Pietzuch, and J. Bacon, "Role-based access control for publish/subscribe middleware architectures", In International Workshop on Distributed Event-based Systems (DEBS), 2003.

[4] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe", ACM Computing Surveys, volume 35, pp. 114-131, 2003.

[5] B. Gerkey and M. Mataric, "Multi-robot task allocation: Analyzing the complexity and optimality of key architectures," In the Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3862-3868, 2003.

[6] B. P. Gerkey and M. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems", International Journal of Robotics Research, volume 23(9), pp. 939-954, 2004.

[7] D. Kidston and I. Labb, "A policy-based resource reservation service for maritime tactical networks", Technical report, Defense Research and Development Canada, 2007.

[8] H. Kuhn, "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly, volume 2, pp. 83-97, 1955.

[9] M. Mataric, G. Sukhatme, and E. Stergaard, "Multi-robot task allocation in uncertain environments", Autonomous Robots, volume 14(2-3), pp. 255-263, 2003.

[10] W. Shu, "Optimal resource allocation on grid computing using a quantum chromosomes genetic algorithm", IET Conference on Wireless, Mobile and Sensor Networks (CCWMSN). pp. 1059-1062, 2007.

[11] M. Younis, K. Akkaya, and A. Kunjithapatham, "Optimization of task allocation in a cluster based sensor network", In the Proceedings of Eighth IEEE International Symposium on Computers and Communication (ISCC), 2003.

[12] Y. Yu and V. Prasanna, "Energy-balanced task allocation for collaborative processing in wireless sensor networks", Mobile Networks and Applications, volume 10 (1-2), pp. 115-131, 2005.

[13] B. Zhao, M. Wang, Z. Shao, and J. Cao, “Topology-Aware Energy Efficient Task Assignment for Collaborative In-Network Processing in Distributed Sensor Systems”, chapter Distributed Embedded Systems: Design, Middleware and Resources, Springer Boston, volume 271, pp. 201-211, 2008.

[14] S. Ahmed, T. Pongthawornkamol, K. Nahrstedt, M. Caesar, and G. Wang, "Topology Aware Optimal Task Allocation for Publish/Subscribe Based Mission Critical Environment,” In the Proceedings of the 28th IEEE conference on Military communications (MILCOM), pp. 2431-2437, 2009.

[15] M. Caleffi, “Mobile Ad Hoc Networks: the DHT paradigm,” In the Proceedings of seventh IEEE International Conference on Pervasive Computing and Communications (PerCom), 2009.

[16] M. Caleffi and L. Paura, “P2P over MANET: Indirect Tree-based Routing,” In the Proceedings of seventh IEEE International Conference on Pervasive Computing and Communications (PerCom), 2009.

[17] J. Eriksson, M. Faloutsos, and S.V. Krishnamurthy, “DART: Dynamic Address RouTing for Scalable Ad Hoc and Mesh Networks,” IEEE/ACM Transactions on Networking (TON), volume 15 (1), pp. 119–132, 2007.

[18] M. Caleffi, G. Ferraiuolo, and L. Paura, “Augmented tree-based routing protocol for scalable ad hoc networks,” In the Proceedings of IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS), pp. 1-6, 2007.

[19] H. Pucha, S. Das, and Y. C. Hu, “Ekta: an efficient DHT substrate for distributed applications in mobile ad hoc networks,” In the Proceedings of Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), pp.163-173, 2004.

[20] T. Zahn and J. Schiller, “MADPastry: A DHT Substrate for Practicably Sized MANETs,” In the Proceedings of fifth Workshop on Applications and Services in Wireless Networks (ASWN), 2005.

[21] F. Delmastro, “From pastry to crossroad: Cross-layer ring overlay for ad hoc networks,” In the Proceedings of Third IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW), pp. 60–64, 2005.

[22] Tutorial for the Network Simulator ns: URL: http://www.isi.edu/nsnam/ns/tutorial/

[23] Publish/Subscribe System: URL: http://en.wikipedia.org/wiki/Publish/subscribe

[24] Haiti Earthquake: URL: goo.gl/0RLaE

[25] Japan Disaster: URL: http://goo.gl/upa8B

[26] Firefighting: URL: goo.gl/2Yyi2

[27] Post-Disaster Rescue: URL: goo.gl/uOrrn

[28] S. Sariel-Talay, T. Balch, and N. Erdogan, "Multiple Traveling Robot Problem: A Solution Based on Dynamic Task Selection and Robust Execution", IEEE/ASME Transactions on Mechatronics, Special Issue on Mechatronics in Multirobot Systems, volume 14(2), pp. 198-206, 2009.

[29] S. Sariel, T. Balch, and N. Erdogan, "Naval Mine Countermeasure Missions: A Distributed, Incremental Multirobot Task Selection Scheme", IEEE Robotics & Automation Magazine, Special Issue on Design, Control, and Applications of Real-World Multirobot Systems, volume 15(1), pp. 45-52, 2008.

[30] S. Sariel, T. Balch, and N. Erdogan, "Robust Multi-Robot Cooperation through Dynamic Task Allocation and Precaution Routines", In the Proceedings of third International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2006, pp. 196-201.

[31] S. Sariel and T. Balch, "Dynamic and Distributed Allocation of Resource Constrained Project Tasks to Robots", Multi-Agent Robotic Systems (MARS) Workshop at the Third International Conference on Informatics in Control, Automation and Robotics, 2006, pp. 34-43.

[32] L. Bertuccelli, H. Choi, P. Cho, and J. How, "'Real-Time Multi-UAV Task Assignment in Dynamic and Uncertain Environments", AIAA Guidance, Navigation, and Control Conference, 2009.

[33] H. Choi, L. Brunet, and J. How, "'Consensus-Based Decentralized Auctions for Robust Task Allocation", IEEE Transactions on Robotics, volume 25(4), pp. 912 - 926, 2009.

[34] F. Araújo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri, "CHR: A Distributed Hash Table for Wireless Ad Hoc Networks", In the Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS), 2005, Columbus, Ohio, USA, June 2005. pp. 407-413.

[35] M. Caramia and S. Giordani, "Resource allocation in grid computing: an economic model", World Scientific and Engineering Academy and Society (WSEAS) Transactions on Computer Research, volume 3(1), pp. 19-27, 2008.

[36] D. Gordon, "The organization of work in social insect colonies", Nature 380:121-124, 1996.

[37] T. Pongthawornkamol, K. Nahrstedt, and G. Wang, "The Analysis of Publish/Subscribe Systems over Mobile Wireless Ad Hoc Networks", In the Proceedings Fourth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous), pp. 1-8, 2007.

APPENDIX A: CODE FOR SHELL SCRIPTS

```
# Script for run-mobile.sh (COTAA)
#!/bin/bash

outdir=boeing_mobile
outname=boeing-out.tr
scdir=scen_out

for nnode in 40 80 120 160 200
do
for nattr in 32
do
for ntask in 1 2 4 8 16 32
do
for spd in 1 5 10 15 20
do
for pt in 50
do
for seed in 3 4 5
do
scfile=$scdir/scen-1000x1000-$nnode-$pt-$spd-$seed
../ns boeing_mobile.tcl $nnode $nattr $ntask $seed $scfile >/dev/null 2>err
rep=`grep "Repair at" err | wc`
grep "^s\|^f" $outname | grep "RTR" | cut -d' ' -f9 > load
sum=`./count ./load`
mv err $outdir/err-$nnode-$ntask-$spd-$pt-$seed
echo $nnode $ntask $spd $pt $seed $sum $rep >> $outdir/result_mobile
done
done
done
done
done
done
```

```
# Script for run-static.sh (COTAA)

#!/bin/bash

outdir=boeing_static
outname=boeing-out.tr

for nnode in 16
do
for nattr in 32
do
for ntask in 1 2 4 8 16
do
for seed in 1 2 3 4 5
do
../ns boeing_mobile.tcl $nnode $nattr $ntask $seed >/dev/null 2>err
comp=`grep "computation" err | cut -d' ' -f5`
comm=`grep "rcvd TASK" err | tail -n 1 | cut -d' ' -f7`
rel=`grep "rcvd TASK" err | wc`
grep "^s\|^f" $outname | grep "RTR" | cut -d' ' -f9 > load
sum=`./count ./load`
mv err $outdir/err-$nnode-$nattr-$ntask-$seed
echo $nnode $nattr $ntask $seed $comp $comm $sum $rel >> $outdir/result_static
done
done
done
done
```

```
# Script for run-static.sh (DOTAA)

#!/bin/bash

outdir=boeing_static_result
outname=boeing-out.tr
scdir=Static
area=10000x10000

for nnode in 4 20 60 80 100 120 140
do
for seed in 3
do
for ntask_per_grp in 2 3 5 10 20 25 30
do

scfile=$scdir/scen-$area-$nnode
../ns boeing_ota.tcl $nnode $ntask_per_grp $seed $scfile >/dev/null 2>output_static
ntask=$(($ntask_per_grp * $nnode/4))

grep "Elapsed" output_static | cut -d ' ' -f3 > time
avg_time=`./average ./time`
mv time $outdir/time-$nnode-$ntask_per_grp
echo $nnode $ntask_per_grp $avg_time >> figs_static/result_time_$area

grep "^s\|^f" $outname | grep "RTR" | cut -d' ' -f9 > load
sum=`./count ./load`
mv load $outdir/load-$nnode-$ntask_per_grp
echo $nnode $ntask_per_grp $sum >> figs_static/result_load_$area

done
done
done
```

Script for run-mobile.sh (DOTAA)

```
#!/bin/bash

outdir=boeing_result
outname=boeing-out.tr
scdir=Mobility
area=1000x1000

for nnode in 4 20 40 60 80 100
do
for pt in 20
do
for spd in 1 5 10 15 20 25 30 35 40 45 50
do
for seed in 3
do
for ntask_per_grp in 2
do

scfile=$scdir/scen-$area-$nnode-$pt-$spd-$seed
../ns boeing_ota.tcl $nnode $ntask_per_grp $seed $scfile >/dev/null 2>output
ntask=$(($ntask_per_grp * $nnode/4))

grep "Elapsed" output | cut -d ' ' -f3 > time
avg_time=`./average ./time`
mv time $outdir/time-$nnode-$ntask-$spd-$pt-$seed
echo $nnode $ntask $spd $pt $seed $avg_time >> figs/result_time_$area

grep "^s\|^f" $outname | grep "RTR" | cut -d' ' -f9 > load
sum=`./count ./load`
mv load $outdir/load-$nnode-$ntask_per_grp-$spd-$pt-$seed
echo $nnode $ntask_per_grp $spd $pt $seed $sum >> figs/result_load_$area

done
done
done
done
done
```