Brunel University
School of Engineering and Design
Electronic and Computer Engineering

Personal Mobile Grids with a Honeybee Inspired Resource Scheduler

by
Heba Abdullataif Kurdi

A thesis submitted for the degree of Doctor of Philosophy

March 2010

Abstract

The overall aim of the thesis has been to introduce Personal Mobile Grids (PM-Grids) as a novel paradigm in grid computing that scales grid infrastructures to mobile devices and extends grid entities to individual personal users. In this thesis, architectural designs as well as simulation models for PM-Grids are developed.

The core of any grid system is its resource scheduler. However, virtually all current conventional grid schedulers do not address the non-clairvoyant scheduling problem, where job information is not available before the end of execution. Therefore, this thesis proposes a honeybee inspired resource scheduling heuristic for PM-Grids (HoPe) incorporating a radical approach to grid resource scheduling to tackle this problem. A detailed design and implementation of HoPe with a decentralised self-management and adaptive policy are initiated.

Among the other main contributions are a comprehensive taxonomy of grid systems as well as a detailed analysis of the honeybee colony and its nectar acquisition process (NAP), from the resource scheduling perspective, which have not been presented in any previous work, to the best of our knowledge.

PM-Grid designs and HoPe implementation were evaluated thoroughly through a strictly controlled empirical evaluation framework with a well-established heuristic in high throughput computing, the opportunistic scheduling heuristic (OSH), as a benchmark algorithm. Comparisons with optimal values and worst bounds are conducted to gain a clear insight into HoPe behaviour, in terms of stability, throughput, turnaround time and speedup, under different running conditions of number of jobs and grid scales.

Experimental results demonstrate the superiority of HoPe performance where it has successfully maintained optimum stability and throughput in more than 95% of the experiments, with HoPe achieving three times better than the OSH under extremely heavy loads. Regarding the turnaround time and speedup, HoPe has effectively achieved less than 50% of the turnaround time incurred by the OSH, while doubling its speedup in more than 60% of the experiments.

These results indicate the potential of both PM-Grids and HoPe in realising futuristic grid visions. Therefore considering the deployment of PM-Grids in real life scenarios and the utilisation of HoPe in other parallel processing and high throughput computing systems are recommended.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I owe my deepest gratitude to my supervisor, Hamed Al-Raweshidy. Professor Al-Raweshidy, has not only taught me how to pursue independent research, but his diligence and commitment to research will also greatly influence me for many years to come. I am grateful for having the opportunity to learn from him and work with him.

I would like to express my sincerest appreciation to my second supervisor, Maozhen Li in recognition of his continuous support, encouragement and patience. Dr. Li. has been a constant source of detailed help regarding all areas of my research.

I am grateful to Dr. Maysem Abbod; his revision and thoughtful comments have been really beneficial in shaping all scheduling-related chapters in my thesis. Thanks to Dr. Ali Mousavi also for helping in queuing theory issues.

I am thankful to all my wonderful lab mates especially Nagham Saeed; many of my research findings arose during discussion with her. Thanks Nagham for the good times. I have really enjoyed working with you. I would also like to thank Carlene Campbell for continuously encouraging and believing in me. Thanks to my colleagues Areeb Al-Owisheq and Areej Al-Wabel for their thoughtful comments.

I acknowledge the Al-Imam Mohammad Ibn Saud Islamic University and the Ministry of Higher education in Saudi Arabia for providing me with a scholarship to pursue this PhD.

The entirety of this work would have never been successful without the help of Almighty ALLAH at first, then the endless support of a very special person, Yousef Abdulghani, my husband. He has made, and is still making, a lot of sacrifices to enable me to develop my academic career. His love has made me capable of weathering all the ups and downs throughout the ordeal of my doctoral study.

I am extremely grateful to my parents, Mum Fareeda Kamel and Dad Abdullataif Kurdi (may Almighty ALLAH bless his soul), in every initiative I take in my life. I hope I have fulfilled a part of their dream.

I would like to thank my children, Majid, Mohannad, Majd, Moath and Mohammad, for their understanding, patience and unconditional love.

Heba Kurdi

January 30, 2010, London, UK

Author's Declaration

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student

List of Abbreviations

3D	Three Dimensional
ABC	Artificial Bee Colony
ACD	Automated Call Distribution
ACO	Ant Colony Optimisation
Akogrimo	Access to Knowledge through the Grid in Mobile World
AmI	Ambient Intelligence
ANOVA	Analysis of Variance
ASP	Application Service Provider
BoT	Bag-of-Tasks
CaaS	Communication as a Service
CHF	Composer Help Flag
CHM	Composer Help Message
CHMD	Composer Help Message Duration
CHT	Composer Help Threshold
CPU	Central Processing Unit
CRM	Customer Relationship Management
CW	Current Workload
DAG	Directed Acyclic Graph
DL	Divisible Workload
DS	Device Score
EaaS	Everything as a Service
EHL	Executer Help List
EHM	Executer Help Message
EHMD	Executer Help Message Duration
EHT	Executer Help Threshold
FIFO	First In First Out
GRACE	Grid Architecture for Computational Economy
GSP	Grid Service Provider
HaaS	Hardware as a Service
HoPe	Honeybee inspired resource scheduling heuristic for Personal Mobile Grids
HPC	High Performance Computing
HTC	High Throughput Computing
IaaS	Infrastructure as a Service
ID	Identification number
IEEE	Institute of Electrical and Electronics Engineers
IrDA	Infrared Data Association

IST	Information Society Technology
JCR	Job Collection Rate
JMX	Java Management Extensions
MAC	Medium Access Control
MAGNT	My Personal Adaptive Personal Global Net
MPI	Message Passing Interface
MR	Medical Record
NAP	Nectar Acquisition Process
NGG	Next Generation Grids
OSH	Opportunistic Scheduling Heuristic
P2P	Peer-to-Peer
PaaS	Platform as a Service
PacWoman	Power Aware Communications for Wireless Optimised Personal Area Networks
PAN	Personal Area Network
PBX	Private Branch Exchange
PDA	Personal Digital Assistant
PDE	Personal Distributed Environment
PG	Personal Grid
PM-Grid	Personal Mobile Grid
PML	Personal Mobile Learning
PM-MR	Personal Mobile Medical Record
PN	Personal Network
PN@home	Personal Networks at Home
PNF	Personal Network Federation
PN-F	Personal Network Federation
PNP	Personal Network Provider
PNP2008	Personal Network Pilot 2008
PSO	Particle Swarm Optimisation
QoS	Quality of Service
RGR	Result Generation Rate
RM	Ready Message
RW	Remaining Workload
SaaS	Software as a Service
SOA	Service Oriented Architecture
TCR	Task Collection Rate
TDT	Tremble Dance Threshold
TT	Turnaround Time
USB	Universal Serial Bus
UWB	Ultra-Wideband
VaaS	Voice as a Service
VO	Virtual Organisation

WDT	Waggle Dance Threshold
WT	Waiting Time
WWRF	Wireless World Research Forum
XaaS	Everything as a Service
ZigBee	Zonal Intercommunication Global-standard, where Battery life was long, which was Economical to deploy, and which exhibited Efficient use of resources

Chapter 1
Introduction

1.1 Introduction

In grid computing [1] a set of computational resources are combined to form a large-scale distributed system in which all resources can be shared. This has the great advantage of providing a resource-rich infrastructure capable of solving data intensive and complex computational problems, such as protein folding and weather forecasting, in an acceptable time and at a reasonable cost.

However, there are two main problems with current grid systems. First of all, they are of very restricted access; they are only available for people in enterprise and research domains. In other words, personal users (individuals outside these domains) are not permitted [2, 3]. Additionally, available grid middleware systems are of very heavy weight in terms of implementations [4]. This is to say, mobile devices cannot be utilised.

Bridging the gap between personal users with mobile devices and grid environments is the end objective of Personal Mobile Grids or simply PM-Grids. This thesis sets out to construct PM-Grids as a new paradigm in grid computing to empower individuals constrained with resource limited devices by providing a ubiquitous resource-rich infrastructure.

Given that Personal Area Networks (PANs) [5, 6] and Personal Networks (PNs) [7, 8] interconnect personal devices, allowing resources such as data, peripherals and secondary storage for sharing, the next logical step is to superimpose grid functionality over these networks offering additional resources such as processors cycles and memories for sharing. Thus, the net result is a virtual supercomputer which can be accessed at anytime and anywhere: a PM-Grid.

However, a very demanding challenging problem becomes apparent when multiple resources are shared. Indeed, it is not a domain specific problem.

Rather, it is a universal optimisation problem that has been subject to extensive research for decades: the resource scheduling problem. When there are multiple machines and a set of jobs, how should machines be allocated to jobs in order to optimise a certain performance measure, such as the job turnaround time or the number of late jobs. This problem, in many of its forms, is known to be NP-complete [9]. It becomes even more complicated and challenging in highly dynamic and unreliable networks [10-12], such as those underlying PM-Grids, due to nodes joining and leaving, switching on and off and working at varying paces.

Therefore, a key to any successful grid system is an efficient scheduler that allocates available resources to incoming jobs. However, conventional grid schedulers are clairvoyant scheduling policies which assume that information about jobs is available to the scheduler before jobs enter the system, in static scheduling, or at least before starting the execution, in dynamic scheduling. Additionally, conventional grid schedulers are usually of centralised and static scheduling policies. A central authority generates a complete schedule prior to execution which other nodes uphold [13]. Such a scheduling scheme severely restricts the scalability of the system. It is prohibitively expensive to generate and simply impractical in many situations where high dynamism is an important issue.

Therefore, a Honeybee inspired resource scheduling heuristic for Personal Mobile Grids (HoPe) is proposed in this thesis with a radical approach to grid scheduling. HoPe implements a non-clairvoyant, self-management and adaptive scheduling policy. In this scheme, no job information is presumed to be available prior to execution, and the scheduling policy is carried locally in each machine based on its perception of the current system state. This approach has its roots in techniques observed in honeybees during their Nectar Acquisition Process (NAP).

This chapter provides a high level overview of the whole thesis. It briefly presents the motivation for the research in section 1.2, then identifies the research overall aim and objectives in section 1.3. Technical challenges are highlighted in section 1.4. The main scientific contributions are presented in section 1.5, while the thesis scope is outlined in section 1.6. The chapter concludes by outlining the structure of this thesis in section 1.7.

1.2 Motivation

The motivation for this thesis is four-fold:

First, the need for grid systems which support the vision of Next Generation Grids (NGG) [14-16] scaling grids to a larger number of entities and smaller devices as well as the vision of Ambient Intelligence (AmI), where humans are surrounded by computing and networking technologies unobtrusively embedded in their surroundings. Current grid architectures and technologies do not meet the requirements for turning these ambitious grid visions into reality [17, 18].

Second, the mobile device market is evolving with a progressive reduction of costs and a continuous improvement in performance, rapidly increasing the number of users and applications of such devices. The Wireless World Research Forum (WWRF) predicts that there will be 1000 wireless devices per person on average in 2017 [19]. One speculates how a personal user will be able to manage such a vast number of devices and efficiently utilise scattered resources among them. It seems reasonable to enable personal users to efficiently share resources including processor cycles, storage capacity and other functionalities among their devices in the form of services available across a global network environment such as computational grids.

Third, people are increasingly keen to frequently replace or upgrade their personal computers to gain more processing power and memory. Sometimes, they need to run complex computational jobs which their desktops or laptops cannot accommodate, or while they are on the move. People are becoming frustrated with the need to move data between their different electronic devices. For instance, a person might have several address books spread over his/her devices. Indeed, there is a need to allow users to harness all processing powers, memory storages and data files scattered across their computing and communication devices, in the form of services available across computational grids, so they can ubiquitously access data and run jobs.

Fourth, colonies of social insects such as bees and ants present an intelligent collective behaviour although they are composed of simple individuals of limited capability. These successful systems from nature have inspired researchers in solving many optimisation problems including the resource scheduling problem. Among all social insects, the technique underlying the NAP in honeybees is the greatest metaphor of efficient cooperation [20]. Exploiting this technique to solve the highly demanding resource scheduling problem in grid computing in particular is an unexplored area, to the best of our knowledge.

1.3 Research Aim and Objectives

The overall aim of the thesis is to introduce PM-Grids as a novel paradigm in grid computing for endowing individuals with resource-rich infrastructures that can serve as general purpose personal mobile and virtual supercomputers. The research aim is addressed through the following objectives:

1. To review the area of grid computing to identify related paradigms to PM-Grids.
2. To introduce PM-Grids to empower personal mobile users with ubiquitous access to their data and computing resources. This objective involves two sub-objectives:
    - 2.1. To develop architectural designs for PM-Grids.
    - 2.2. To build simulation models for PM-Grids.
3. To review the area of resource scheduling to identify required features for an efficient resource scheduler in PM-Grid environments.
4. To develop a resource scheduling heuristic to efficiently schedule PM-Grid resources. This objective involves two sub-objectives:
    - 4.1. To design the heuristic.
    - 4.2. To implement the heuristic.
5. To empirically validate the PM-Grid models using the developed scheduling heuristic and to analyse the results.

1.4 Challenges

There are many technical challenges in developing PM-Grids. These challenges are inherited from the original components of PM-Grids in three fields, as illustrated in Figure 1.1:

- Grid computing: Grid computing is a rapidly developing area of research, with heavy implementations which support neither mobile nor personal users.
- Personal Networks: PNs are a relatively new area of research with demanding issues such as unreliable connectivity, heterogeneity in terms of hardware and software, and high security risks.
- Mobile computing: Mobile computing is a challenging research area which needs to tackle problems such as resource limitation of mobile devices, low bandwidth and high dynamism.

These challenges shape the development of PM-Grids more demanding than with other grids.

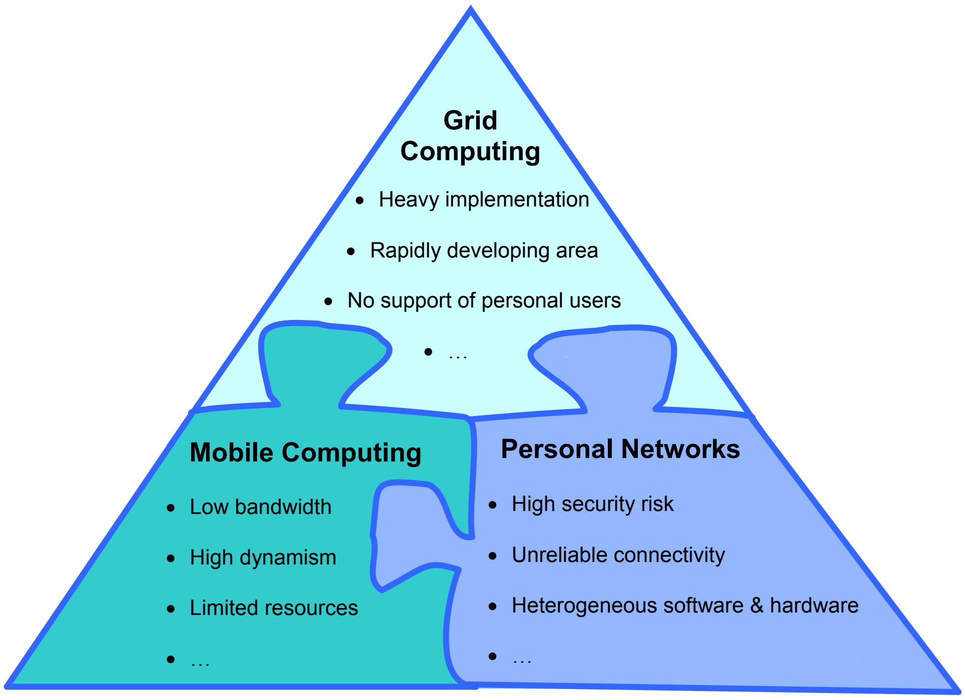


Figure 1.1: PM-Grids Challenges

1.5 Main Contributions

There are seven main contributions of this thesis which are summarised in the following sections.

1.5.1 Architectural Designs and Models for PM-Grids

As indicated in section 1.2, there are gaps between current grids and the visions of future grids. Neither traditional grid architectures nor vast extensions to them can satisfy the requirements of the NGG; the way forward is to design an architecture based on the properties of NGG and implement it [17].

Therefore, this thesis has originated designs for PM-Grids based on the PNs architecture and as a natural extension to them, given that the NGG features have been explicitly addressed in their design. An abstract layered view, a detailed inside view and simulated models at different scales in terms of number of devices per cluster, are presented and evaluated in this thesis.

1.5.2 Detailed Design and Implementation of HoPe

The extremely dynamic nature, diversity and limited capabilities of resources, as well as difficulties in predicting the nature and timing of incoming jobs (non-clairvoyant scheduling), are all factors which considerably influence the complexity of the scheduling problem in PM-Grids. Through observation, the honeybee colony solves an extraordinarily difficult scheduling problem while allocating bees to nectar sources in nature, through a simple decentralised cooperative and adaptive self-scheduling policy.

This observation has inspired this thesis to follow a similar approach in scheduling PM-Grid resources. A detailed design, implementation and evaluation of HoPe are initiated in this thesis. To the best of our knowledge, HoPe is the first algorithm to shed light on the non-clairvoyant scheduling problem in grid computing. It is the first honeybee-inspired algorithm attempting to solve the resource scheduling problem relying totally on local and easily calculated parameters which is considered among the most important features of the honeybee colony [20].

1.5.3 Detailed Analysis of the NAP

Honeybees present an intelligent collective scheduling behaviour while allocating themselves to nectar sources under extraordinarily difficult conditions during the NAP [20]. This has motivated some previous work to analyse and model the honeybee colony and its NAP. However, these are concrete mathematical and probabilistic models quantifying features of the honeybee behaviour based on certain sets of predefined assumptions. The problem with this approach is that the honeybee colony, as in the case of all biological systems, has unique characteristics that are apparently different from the mathematical assumptions that lie beneath the analytical models.

Therefore, this thesis has initiated a queuing theory with a simulation based approach to NAP modelling from the resource scheduling perspective. A generic model for the NAP is developed as a queuing network which is simulated in several representative scenarios. Additionally, detailed algorithmic analysis and modelling based on honeybee techniques are presented. Some of these techniques have not been considered in previous work, namely, the tremble dance that controls the nectar influx to the hive and the dependence only on locally calculated parameters.

1.5.4 Comprehensive Taxonomy of Grid Systems

Despite rapid developments in grid computing, there has been, surprisingly, no research into reviewing or classifying newly emerged grid systems. A survey and a classification scheme for emerging grids are initiated, in this thesis, to bridge this gap. This classification is extended in the form of a comprehensive taxonomy for both emerging and traditional grids which is significant for the following reasons. First, it facilitates studying grid systems under one framework. Second, it allows one to see the main design features of grid systems clearly and assists a detailed comparison between them. Third, it helps in understanding current research trends in grid computing and anticipating future trends. Fourth it provides a common set of terminologies for grid systems in an attempt to establish a solid framework for the rapidly evolving area of grid computing.

1.5.5 Unified Framework for Resource Schedulers

In contrast to the scarcity of resources involved in proposing taxonomies for grid systems, a plethora of literature has proposed taxonomies for resource schedulers in distributed systems, in general, and grid computing, in particular. This abundance of taxonomies has resulted in scattered nomenclatures as well as vague and inconsistent terminologies in the literature, necessitating the development of a unified view of the previous work.

Therefore this thesis presents a common framework for resource scheduling with a unified presentation of previously published taxonomies, indicating the different terminologies in use. The intention has been to provide a means to help in designing and analysing resource schedulers and also in comparing them. Such a framework is deemed necessary to amalgamate the area of resources scheduling under a common, uniform set of nomenclatures and terminologies.

1.5.6 Controlled Empirical Evaluation Framework

A controlled empirical evaluation framework to prove the concept of PM-Grids and to evaluate the performance of HoPe is developed in this thesis. A flexible simulator is built for this purpose, allowing the control of experimental parameters (job interarrival time and number of devices per cluster), randomising extraneous variables (processor capacity and job size) as well as measuring and analysing various performance metrics (stability, throughput and turnaround time). An optimum value, worst bound and a benchmark algorithm (the Opportunistic Scheduling Heuristic – OSH) are employed to assess HoPe performance.

1.5.7 Performance Models of HoPe and OSH Behaviours

Mathematical performance models are generated, using multiple regressions and quadric equations, to predict the performance of HoPe and OSH in regard to stability, net throughput, turnaround time and speedup. In addition, a three dimensional (3D) graphical model is created for each predicted mathematical model. The statistical significance of models predicted is evaluated by the analysis of variance (ANOVA) test which determines which factors significantly affect the performance metric in the study. These models assist in gaining clearer insight into the behaviour of each heuristic under various running conditions of job interarrival times and grid scales.

1.6 Thesis Scope

This thesis contemplates novel paradigms from different areas resulting in research of a multidisciplinary nature that involves cross-fertilisation of ideas from grid computing, mobile computing and networking among others. Therefore, it was necessary to outline a clear scope to successfully accomplish the objectives in the given time frame.

First, the scope in terms of grid computing, it should be noted that although building computational grids involves several issues, this thesis has only considered the resource scheduling issue as resource schedulers are the heart of any grid system.

Second, in terms of underlying networks, this thesis has considered PNs as the basic infrastructure for PM-Grids as they have the potential for realising the NGG vision. Investigating other networks and infrastructures is considered beyond the thesis scope.

Third, in terms of mobile computing, while the word “mobile” is stressed throughout this thesis, the main concern is the highly dynamic nature of the mobile devices environment and their limited resources in terms of processor capacity in particular, rather than the usual issues raised with mobility such as code migration, battery life time and limited bandwidth. The thesis scope is summarised in Figure 1.2. Examples in each research domain given in the figure are not exhaustive.

It is important to note that although some grid practitioners restrict the term “grid” to computational environments which span multiple administration domains [1], this thesis has utilised the term grid to refer to the proposed environment, PM-Grid, which does not necessarily span multiple administration domains. This aligns with the approach followed by some leading grid authorities such as Sun Microsystems [22].

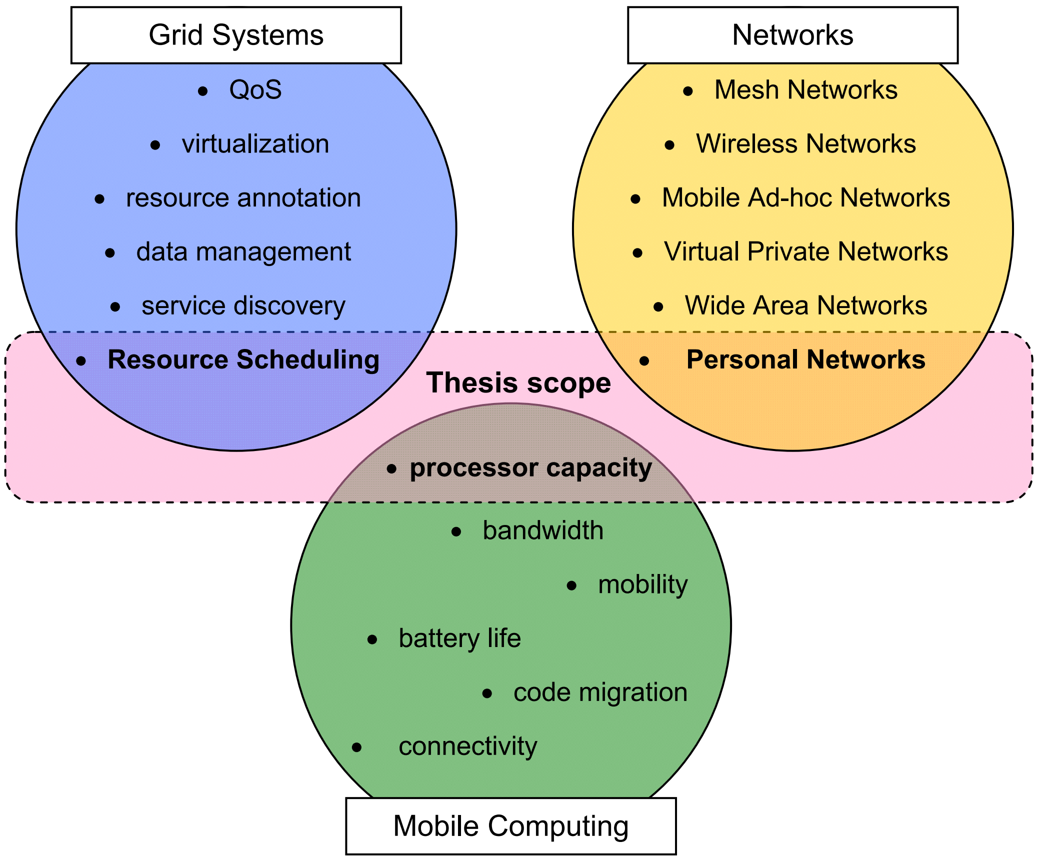


Figure 1.2: Thesis Scope

1.7 Thesis Outline

The work presented in this thesis is organised into seven chapters. Each chapter starts with a brief introduction highlighting the main contributions and providing an overview of that chapter. At the end of each chapter, brief concluding remarks and a list of references are presented. Figure 1.3 illustrates the structure of the thesis and relationship to the thesis objectives presented in section 1.3.

The next six chapters contain more detailed information about the theoretical background and technical development of PM-Grids:

Chapter 2 presents a detailed background about grid computing and its evolution over the last few years. It defines traditional and emerging grids and provides a skeletal classification for the latter which is extended into a comprehensive taxonomy for both traditional and emerging grids. The chapter reviews and compares related paradigms to grid computing. The chapter concludes with a brief summary and discussion.

Chapter 3 introduces the PM-Grid concept by defining PM-Grids and outlining their potential application areas. It provides a brief background about PANs and PNs then reviews the architectural design of PNs on which the PM-Grid architectural design is based. An abstract layered architecture and a detailed inside view for PM-Grids are illustrated. The chapter concludes with a comparison of the PM-Grid with related work in distributed systems.

Chapter 4 lays the background for HoPe by presenting the resource scheduling problem and its evolution over more than fifty years. It proposes a framework for resource scheduling systems with a unified taxonomy of previous work in the area. The resource scheduling problem in grid environments in particular and its challenges is highlighted with a brief review of three well established grid resource schedulers: Condor [23], Legion [24] and Nimrod-G [25] based on the proposed framework. The chapter concludes with a brief discussion and open research issues.

Chapter 5 details the HoPe analysis and design phases. It starts by articulating the resource scheduling problem in PM-Grids and identifying scheduler requirements to tackle such a challenging problem. It states the broad HoPe hypothesis and discusses it. A detailed analysis of the behaviour of honeybees during the NAP, with algorithmic style and from the queuing theory perspective, is presented. The design and implementation elements of HoPe are identified with the honeybee to PM-Grids and the NAP to HoPe analogies, explained.

Chapter 6 describes in detail the evaluation process, defining the objectives and the experimental design. HoPe and the OSH are analysed and contrasted using the resource scheduling framework, proposed in Chapter 4. PM-Grid simulated models are presented then experimental results are illustrated, analysed and discussed. HoPe and the OSH performance models are generated and discussed.

Finally, Chapter 7 summarises the thesis aims, major contributions and significant findings. It highlights areas and directions for further research.

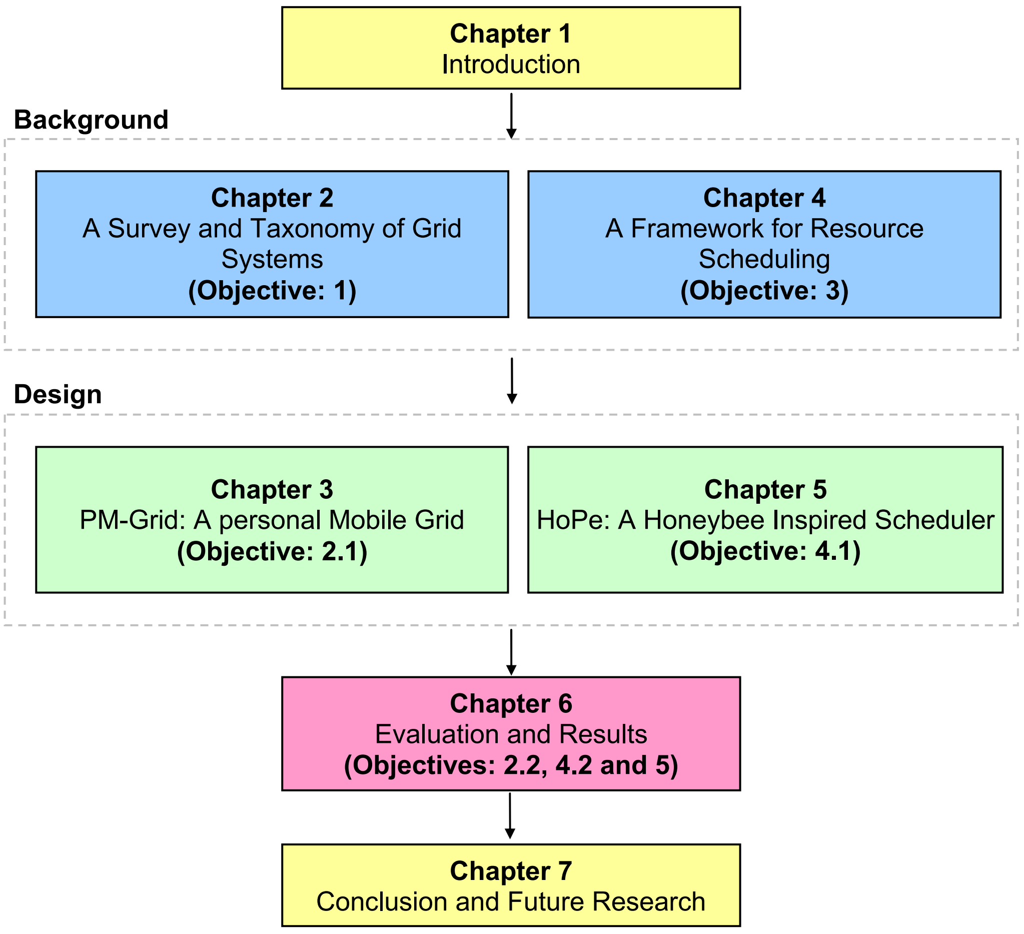

Chapter 1
Introduction
Background
Chapter 2
A Survey and Taxonomy of Grid Systems
(Objective: 1)
Chapter 4
A Framework for Resource Scheduling
(Objective: 3)
Design
Chapter 3
PM-Grid: A personal Mobile Grid
(Objective: 2.1)
Chapter 5
HoPe: A Honeybee Inspired Scheduler
(Objective: 4.1)
Chapter 6
Evaluation and Results
(Objectives: 2.2, 4.2 and 5)
Chapter 7
Conclusion and Future Research

Figure 1.3: Thesis Structure

1.8 References

[1] I. Foster and C. Kesselman, Eds., *The Grid2: Blueprint for a Future Computing Infrastructure*. San Francisco: Morgan Kaufmann, 2003.

[2] J. Han and D. Park, "A lightweight personal grid using a supernode network," in *Proc. 3rd Int. Conf. P2P2003*, pp. 168-175.

[3] K. Amin, G. V. Laszewski and A. R. Mikler, "Grid computing for the masses: An overview," in *Proc. GCC2003,* pp. 164-173.

[4] D. Millard, A. Woukeu, F. Tao, and H. C. Davis, "The potential of grid for mobile e-learning (Poster)," presented at the 4th World Conference on Mobile Learning, Cape Town, South Africa, 2005.

[5] IEEE 802.15 Working Group for WPAN [online]. Available: http://ieee802.org/15/, [accessed Feb. 2, 2010].

[6] R. C. Braley, Ian C. Gifford, and Robert F. Heile, "Wireless personal area networks: an overview of the IEEE P802.15 working group," *SIGMOBILE Mobile Computing Commun. Rev.*, vol. 4, pp. 26-33, 2000.

[7] My Personal Adaptive Global NET (MAGNET) (IST 507102) [online]. Available: http://www.ist-magnet.org, [accessed Feb. 2, 2010].

[8] IST.MAGNET Beyond (IST-FP6-IP-027369) [online]. Available: http://www.magnet.aau.dk, [accessed Feb. 2, 2010].

[9] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Eds., *Handbook on Scheduling: From Theories to Applications.* New York: Springer, 2007.

[10] M. Mitzenmacher, "How useful is old information?" *IEEE Trans. Parallel Distribut. Syst.*, vol. 11, pp. 6–20, 2000.

[11] S. Irani and Y. Rabani, "On the value of coordination in distributed decision making," *SIAM J. Comput.*, vol. 25, no. 3, pp. 498- 519, 1996.

[12] S. Georgiades, M. Mavronicolas and P. Spirakis, "Optimal, distributed decision-making: The case of no communication," in *Proc. Int. Symp. Fundamentals Comput. Theory*, 1999, pp. 293–303.

[13] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "Self-organizing scheduling on the organic grid," *Int. J. of High Performance Comput. Applicat.,* vol. 20, pp. 115–130, 2006.

[14] Expert Group, "Next generation grids: European grid research 2005-2010," Expert Group Rep., Jun. 2003 [online]. Available: ftp://ftp.cordis.lu/pub/ist/docs/ ngg_eg_final.pdf, [accessed Feb. 2, 2010].

[15] Expert Group, "Next generation grids2: Requirements and options for European grids research 2005-2010 and beyond," Expert Group Rep., Jul. 2004 [online]. Available: http://www.semanticgrid.org/docs/ngg2_eg_final.pdf, [accessed Feb. 2, 2010].

[16] Expert Group Final, "Future for European grids: Grids and service oriented knowledge utility," Expert Group Final Rep., Jan. 2006 [online].

Available: ftp://ftp.cordis.europa.eu/pub/ist/docs/grids/ngg3_eg_final.pdf, [accessed Feb. 2, 2010].

[17] K.G. Jeffery, “Next generation grids for environmental science,” *Environmental Modelling & Softw.,* vol. 22, no. 3, pp. 281–287, 2007.

[18] Sajjad, H. Jameel, U. Kalim, S. Han, Y. Lee and S. Lee, “AutoMAGI - an autonomic middleware for enabling mobile access to grid infrastructure,” in *Proc. 2005 ICAS-ICNS,* pp. 73-79.

[19] Jefferies, N., Global Vision for a Wireless World, Wireless World Research Forum, 18th WWRF meeting, Helsinki, Finland, Jun. 2007.

[20] T. D. Seeley, *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies*. MA: Harvard University Press, 1995.

[21] IBM [online]. Available: http://www.ibm.com/us/, [accessed Feb. 2, 2010].

[22] Sun Microsystems [online]. Available: http://www.sun.com/, [accessed Feb. 2, 2010].

[23] Condor Project [online]. Available: http://www.cs.wisc.edu/condor, [accessed Feb. 2, 2010].

[24] Legion: A Worldwide Virtual Computer [online]. Available: http://legion.virginia.edu/, [accessed Feb. 2, 2010].

[25] DSTC Nimrod/G [online]. Available: http://www.csse.monash.edu/~sgaric/nimrod/, [accessed Feb. 2, 2010].

Chapter 2
A Survey and Taxonomy of Grid Systems

2.1 Introduction

During the last few years, information technology has witnessed a rapid advance in every aspect, including speed and performance. This substantial advancement has affected not only the application areas in which grid technologies can be applied, but also the underlying architecture of how grids are developed, deployed and run. As a result new grid systems have emerged creating a significant evolution in grid systems.

Such advances in information technologies have also evolved new distributed system paradigms, such as utility computing, everything as a service and cloud computing with similar visions to grid computing. This has raised the question of whether these advances really propose new solutions replacing grid systems, or are merely new commercial names for grid computing.

This chapter includes two main contributions. The first is a survey and a classification scheme of existing state-of-the-art emerging grid systems. Such a survey and classification has not been reported in the literature before, and its importance is to highlight the salient design features of emerging grid systems and to assist in detailed comparisons between them. It helps in understanding current research in grid computing and in anticipating future trends.

The second contribution of this chapter is a comprehensive taxonomy of both traditional and emerging grids. Some earlier works have included simple classifications of traditional grid systems, and the taxonomy presented here agrees with the nature of such classifications. However a large number of additional fundamental distinguishing features are included that have not been presented in any previous work, to the best of our knowledge. Such a comprehensive taxonomy is important to differentiate between grids and facilitate the study of them under one framework. The aim is to provide a common terminology and classification mechanism for grid systems in an attempt to contain the area under one scheme.

In section 2.2 grid computing is defined while its evolution over the last few years is described in section 2.3. Section 2.4 presents a classification of emerging grids which is extended to a comprehensive taxonomy to cover traditional grids in section 2.5. Section 2.6 presents an overview of emerging paradigms related to grid computing, and compares them. Section 2.7 summarises and concludes the chapter.

2.2 Grid Computing

Basically, Grid computing [1] is a relatively new distributed system paradigm where computational resources are coupled together to form a large-scale distributed system where all resources are available for sharing. This has the great advantage of providing a resource-rich infrastructure capable of solving data intensive and complex computational problems such as protein folding and weather forecasting in an acceptable time and at a reasonable cost.

Indeed, there are as many definitions to the grid as the growing number of organisations utilising it. A common theme underlying these definitions is the coordinated resource sharing and problem solving in a Virtual Organisation (VO). A VO is a dynamic set of participants defined around a set of resource sharing rules and conditions as shown in Figure 2.1.

Some grid definitions add additional criteria requiring the grid resources to be distributed across multiple administrative domains [2, 3] or to be geographically distributed [4]. These additional criteria exclude from the grid definition, clusters where shared resources are usually in the same locality and administrative domain. However, some leading grid authorities, such as Sun Microsystems [5], do consider clusters as grid environments, using the term ‘Cluster Grids’ to refer to them [6].

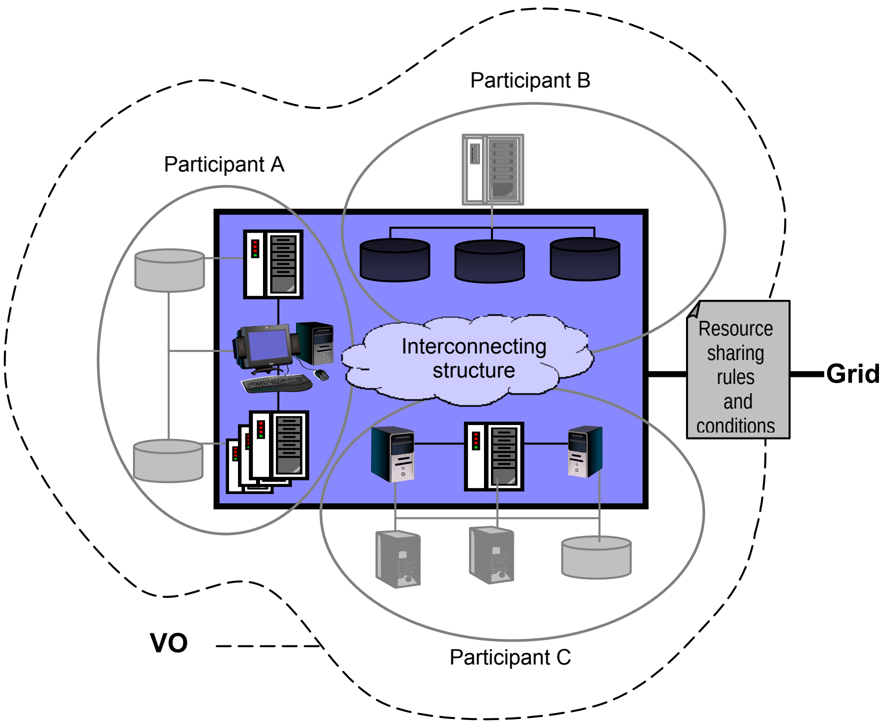

Participant B
Participant A
Interconnecting structure
Resource sharing rules and conditions
Grid
VO
Participant C

Figure 2.1: Grid Environment

2.3 Grid Generations

Grid computing is a rapidly evolving area of research characterised by a number of distinct phases or generations, as shown in Figure 2.2. The grid started in the early nineties, as a model of meta-computing where resources in supercomputers were shared; subsequently the ability to share data was added. These are usually referred to as first generation grids. By the late nineties (1998), the framework was published for second generation grids, which are characterised by their focus on the use of grid middleware systems to glue different grid technologies [7]. In the early millennium (2001) fast data transfer and storage request brokers for persistent data storage with metadata description were added to grid platforms introducing what are usually known as the 2.5 grid generation. Late in 2002, third generation grids originated by combining the Web technology with the second generation grids [8].

Recently, the Next Generation Grids (NGG) [9-11] vision has been defined by a group of independent experts from the European Commission to identify potential European grid research priorities for 2010 and beyond. The NGG vision stresses the necessity for grids to support and extend the Ambient Intelligence (AmI) vision, where humans are surrounded by computing technologies unobtrusively embedded in their surroundings.

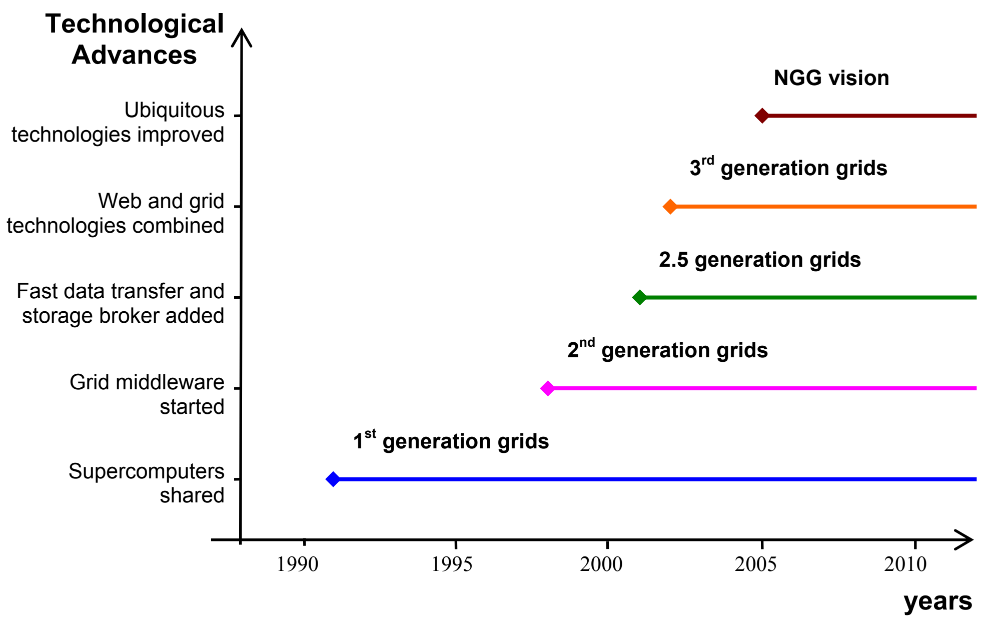


Figure 2.2: Grid Generations

2.4 Features of Next Generation Grids

The prospective NGG vision places scalability, openness to wider user community, pervasiveness and ubiquity, transparency and person-centricity, as its top priorities [9-11]. These are composite features that comprise four main primitive system design features:

- Accessibility: In this context, accessibility means making grid resources available regardless of the physical capabilities and geographical locations of access devices.
- Interactivity: In this context, interactivity refers to the ability of a grid to timely respond to real-time systems requiring rapid response times and synchronous communication.
- User-centricity: A design philosophy in which the needs and expectations of the end user of an interface are the centre of focus.

- Manageability: The ability of a system to automatically manage, adapt, monitor, diagnose and fix itself. A manageable system has embedded intelligent control into its infrastructure to automate its management procedure.

In this thesis, these four features are considered as the main drivers for emerging grids. For simplicity, the term "traditional grids" is utilised to refer to the grid systems that lack the above four features, while the term "emerging grids" refers to recent grid projects that explicitly address at least one of these features.

2.5 Classification of Emerging Grids

There is a fundamental gap between current grid implementations and the prospective NGG vision [8]. However, new grid systems are rapidly emerging with the potential to plug this gap. Surprisingly, no review or classification of emerging grids is available, to the best of our knowledge. Therefore this section sets out to bridge this gap by providing a skeletal classification and a brief survey of emerging grids; more details about each grid category being available in section 2.6. The aim is to give a broad view of the amount and type of work which has been done with respect to each feature specifically, and towards the NGG vision in general, which may drive further research in this area. The classification places emerging grids in groups according to a set of salient features. This allows a convenient means of quickly describing the central aspects of a particular approach, as well as a basis for comparisons between the groups.

After reviewing grid projects and literatures, emerging grids are identified and placed into four main groups, based on the NGG features: Accessible Grids, User-Centric Grids, Interactive Grids and Manageable Grids, as shown in Table 2.1. Each group is divided further into sub-groups based on the most apparent feature that distinguishes it from traditional grids. The table also gives example projects of each emerging grid. However, since the main concern of this section is to survey and classify emerging grids rather than comprehensively reviewing all available projects, the example projects are not exhaustive, but comprehensive enough to cover all the features of the emerging grid which they represent. The scope of each grid is narrowed to concentrate on one feature per category. This is the reason why names such as Ubiquitous or Pervasive Grids, are not used; since these names involve the supporting of combinations of features such as accessibility, interactivity and user-centricity.

However, it should be noted that this classification is not disjoint. This means that a grid system can be classified under more than one of the classification features, for instance a grid G might be mobile, personal, autonomic and interactive at the same time.

Table 2.1: Classification of Emerging Grids

Classification features	Categories	Sub-categories	Main difference from traditional grids	Example projects
Accessibility	Accessible Grids	Ad-hoc Grids	Have no predefined entry points	OurGrid [12]and myGrid [13]
		Wireless Grids	Support wireless connections between grid nodes and interfaces	Innovaticus [14] and FWGrid [15]
		Mobile Grids	Support mobility of clients, services or both	Akogrimo [16]
Interactivity	Interactive Grids	Direct Interactive Grids	Support direct real time interaction with end users	CrossGrid [17]and edutain@grid [18]
		Context-Aware Grids	Interact with the surroundings to build the context and adapt their behaviours	CONTEXT [19]
User-centricity	User-Centric Grids	Customisable Grids	Implement highly personalisable grid portals	MyGrid [13]and Akogrimo [16]
		Personal Grids	Owned or utilised by individuals	Personal Grid [20] and PG [21]
Manageability	Manageable Grids	Autonomic Grids	Utilise ideas from human body's autonomic nervous system to support self-management	IBM OptimalGrid [22] and AutoMAGI [23]
		Knowledge Grids	Utilise knowledge technologies to support self- management	OntoGrid [24]and InteliGrid [25]
		Organic Grids	Utilise ideas from biological systems such as ant or bee colonies to support self- management	Organic Grid [26]

2.6 Taxonomy of Grid Systems

In this section a comprehensive taxonomy of both traditional and emerging grids is proposed based on six nomenclatures. Earlier works [27, 28] have included classifications of traditional grid systems, based on the type of provided solution and the scope of the VO. The taxonomy presented here is consistent with the nature of such classifications and adds four additional fundamental disguising features to better differentiate between grid systems.

The aim is to provide a common terminology and classification mechanism for grid systems in an attempt to connect the area under one scheme.

Figure 2.3 presents the proposed comprehensive taxonomy which categorises grid systems based on six nomenclatures: solution type, VO scope, accessibility, user-centricity, interactivity and manageability. The following sections overview each category.

It important to note that a grid system is to be classified based on the six nomenclatures. For instance, a grid G_1 can be classified as: enterprise, data, closed, batch, organisational and centralised, while a grid G_2 is: personal, computational, wireless, context-aware, user-centric and organic. It is important to consider that some sub-categories can be classified under more than one nomenclature. For instance, Personal Grids can be classified according to the size of the VO and also based on user-centricity. To avoid duplication, each sub-category is classified only under one nomenclature in the taxonomy, but a note is made as necessary regarding where else the sub-category can be classified.

2.6.1 Grids Classified by Solution Type

Grid systems are constructed for different objectives and provide different types of solutions. Based on the type of the provided solution, grid systems are classified as Computational Grids, Data Grids, Service Grids and Access Grids as shown in Figure 2. 3.

2.6.1.1 Computational Grids

Computational Grids are constructed out of highly aggregated computational resources to jointly solve computationally intensive problems that require a great number of CPU cycles. The main solutions offered by these types of grids are the CPU cycles. Computational Grids are further classified based on the main type of hardware resources deployed as:

- Desktop Grids, where desktop computers constitute a considerable volume of grid resources.

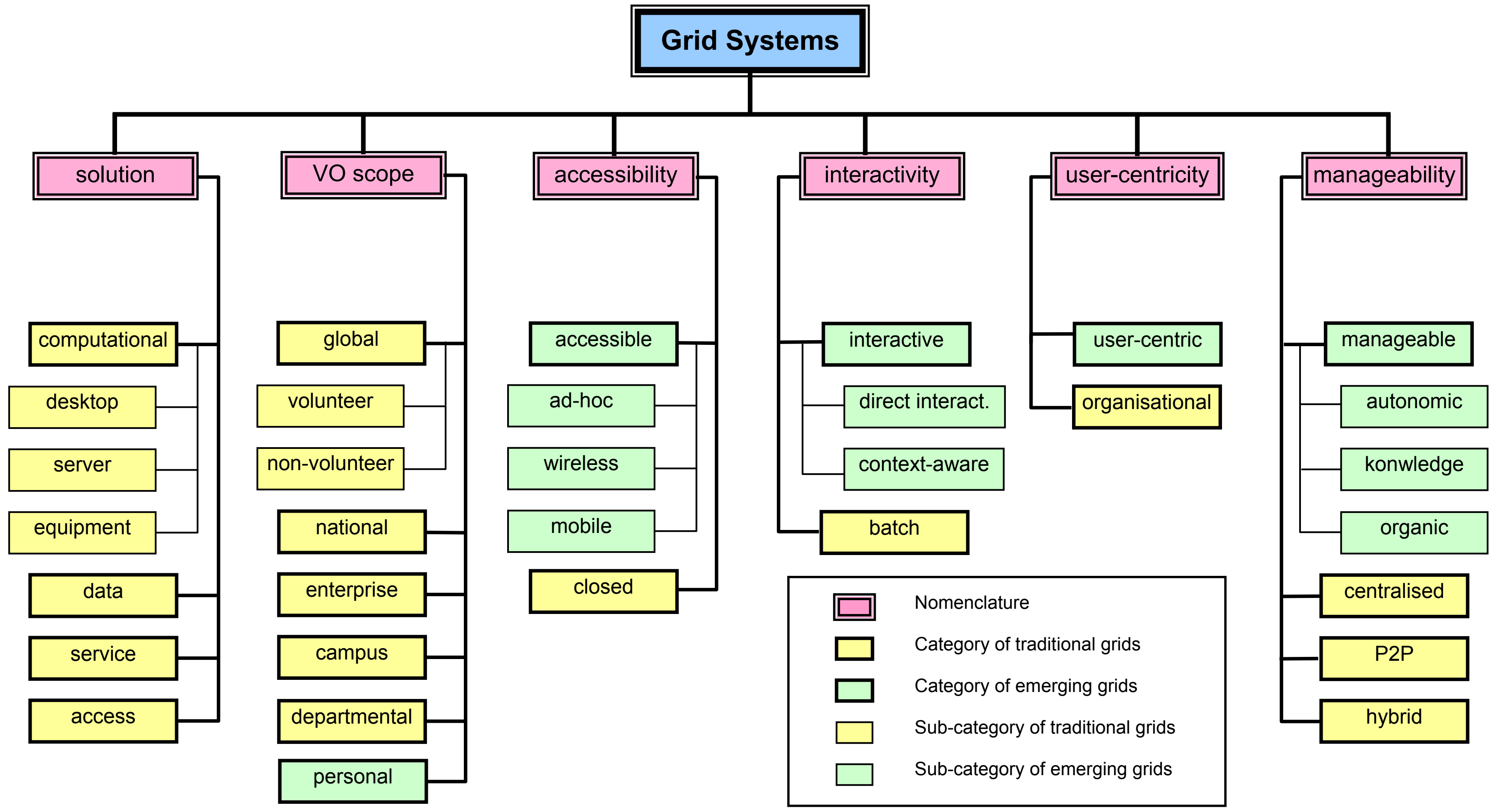


Figure 2.3: Comprehensive Taxonomy of Grid Systems

- Server Grids where resources are usually limited to those available within servers.
- Equipment Grid, or Instrument Grid, includes a key piece of equipment such as a telescope. The surrounding grid, a group of electronic devices connected to the equipment, is used to control the equipment remotely and to analyse the data produced. For instance, in the World-Wide Telescope [1], data from hundreds of individual telescopes all over the world is analysed and categorised using grid technologies to find new phenomena.

2.6.1.2 Data Grids

Data Grids are grid systems in which the main solution offered is storage devices. They are used to provide an infrastructure for accessing, storing and synchronising data from huge distributed data repositories, such as digital libraries or data warehouses, and distributed data-intensive applications such as data mining. Although Data Grids share similarities with other distributed data-intensive paradigms, such as content delivery networks, P2P networks and distributed databases, they are differentiated by heavy computational requirements, wider heterogeneity, autonomy and the concept of VOs. In [29] a detailed taxonomy of data grids is presented which classifies Data Grids based on grid organisation, transport technologies used, grid environments and resource allocation and scheduling schemes.

2.6.1.3 Service Grids

Service Grids, also known as Utility Grids, provide commercial computer services such as CPU cycles and disk storage, which can be purchased on demand. They focus on users' satisfaction by combining and delivering services based on their needs and requirements. Grid users send their service requirements together with preference parameters such as Quality of Service (QoS) requirements and cost, to a Grid Service Provider (GSP) or broker who dynamically allocate them appropriate grid middleware services.

2.6.1.4 Access Grids

Access Grids consist of distributed input and output devices, such as speakers, microphones, video cameras, printers and projectors connected to a grid. Hence these devices provide multiple access points to the grid where clients can issue requests and receive results in large-scale distributed meetings and training sessions. Access grids aim at group-to-group collaboration in high-end workspaces by providing virtual rooms maintaining project-related applications which are available to all project members [1].

Recent trends foreshadow the incorporation of wireless and mobile devices into grid systems. In this case wireless and mobile devices can serve as entry points to the grid where grid users can gain access to grid resources. Theses grids are known as Wireless Access Grid or a Mobile Access Grid. More details about Wireless Access Grids and Mobile Access Grids are presented in section 2.6.3.2.2 and Section 2.6.3.2.3 respectively.

2.6.2 Grids Classified by Virtual Organisation Scope

Figure 2.3 shows that, according to the size or scope of the underlying VO, grids are classified into Global Grids, National Grids, Enterprise Grids, Cluster Grids (Campus Grids and Departmental Grids) and Personal Grids.

2.6.2.1 Global Grids

Global Grids are established over the Internet to provide individuals or organisations with grid power anywhere in the world. This type of grid is usually referred to as Internet Computing as well. Global Grids consist of a collection of smaller grids and other geographically distributed resources with agreed upon global usage policies and protocols to enable resource sharing. Global Grids can be further classified into:

- Volunteer Grids which offer an efficient solution for distributed computing. They allow Internet users to contribute their unused computer resources, to collectively accomplish non-profit, complex scientific computer-based tasks. Consumption of resources is strictly limited to the controlling organisation or application.
- Non-Volunteer Grids which contain dedicated machines only and clear pre-defined pricing and usage schemes.

2.6.2.2 National Grids

National Grids are restricted to the computer resources available within one country's borders. They are only available to organisations of national importance and are usually funded by governments. Many countries are establishing National Grid projects to provide a common Infrastructure for e-science. Europe has established itself as the world leader in the field by investing heavily in grid computing programmes at both the national and the European levels [30].

2.6.2.3 Enterprise Grids

An Enterprise Grid is concerned with using idle desktop resources within an enterprise [31]. It is managed by a single organisation, i.e. the enterprise, and available only to its users [32]. However, Enterprise Grids can also be deployed within large corporations that have a global presence [33].

2.6.2.4 Intra-Grids

The term Intra-Grid (Cluster Grid) is used to refer to two types of grids: Campus Grids and Departmental Grids. In Campus Grids resources are restricted to those available within a single organisation. They are only accessible by members of the host organisation. Departmental Grids are even more restricted than Campus Grids. They are only available for people within a department boundary.

Campus and Departmental Grids sometimes are not considered as true grid environments as they do not span multiple administrative domains and are not geographically distributed. There are more relevant to cluster, than to grid computing. Nevertheless, as indicated earlier, some leading grid authorities such as Sun Microsystems [5] do consider Cluster Grids as true grid environments.

2.6.2.5 Personal Grids

Traditional grids are designed specifically for people involved in research and large industry domains. Hence, it is difficult for personal users, individuals outside these domains, to utilise or construct a grid system for their own needs [20]. Therefore, Personal Grids are emerging to bridge this gap.

Personal Grids are grid systems with the most limited scope of the VO. They are owned, constructed, and managed by owners and other people whom they trust. Research into Personal Grids is still at a very early stage. A framework for a Personal Grid that consists of a set of networked personal desktop computers is proposed in [20]. In [21, 34, 35] a Personal Grid (PG) is proposed to allow integrating desktop computers into a virtual server in the Internet. This work is still running; with no implementation details or evaluation available until now. As indicated earlier, Personal Grids can also be classified under User-Centric Grids.

2.6.3 Grids Classified by Accessibility

As shown in Figure 2.3, when accessibility is considered, grids can be classified into two main groups: Closed Grids and Accessible Grids.

2.6.3.1 Closed Grids

Closed Grids are traditional grid environments in which grid nodes are usually stationary with predefined wired infrastructure. Access to grids is allowed only through static predefined entry points.

2.6.3.2 Accessible Grids

The highly structured networks of supercomputers and high performance workstations that dominate traditional grids do not provide ubiquitous accessibility. Hence Wireless Grids, Mobile Grids and Ad hoc Grids have emerged. “Accessible Grids” is an umbrella term employed to refer to these types, as shown in Table 2.1. An Accessible Grid is a grid that might consist of a group of mobile or fixed devices with wired or wireless connectivity and predefined or ad hoc infrastructures.

The main characteristic of an Accessible Grid is its highly dynamic nature which results from the frequently changing structure of underlying networks and VOs due to nodes switching on and off, nodes entering and leaving, nodes’ mobility etc. This is why traditional service discovery, management and security mechanisms may not be optimal for Accessible Grids.

Accessible Grids can be accessed from more geographical locations and social settings than traditional grids. This opens the door for new applications in emergency communication, disaster and battlefield management, e-learning and e-healthcare among others.

One of the most critical issues in understanding Accessible Grids is having an accurate definition, or at least determination, of each type. However, researchers offer no consistent definition of any of the three terms. Wireless Grids emphasise the wireless connectivity, Ad hoc Grids stress the ad hoc nature of VOs, while Mobile Grids focus on the mobility related issues such as job migration and data replication.

2.6.3.2.1 Ad hoc Grids

Although the ad hoc and sporadic nature of grids was observed within the first documented Globus [36] Grid application, traditional grids fail to support certain aspects of ad hoc environments [37], such as constantly changing membership with a lack of structured communications infrastructure. As a result, Ad hoc Grids have emerged.

An Ad hoc Grid is a spontaneous formation of cooperating heterogeneous computing nodes into a logical community without a pre-configured fixed infrastructure and with minimal administrative requirements [38], as shown in Figure 2.4. Thus, the traditional static grid infrastructure is extended to encompass dynamic additions with no requirements of formal, well-defined or agreed grid entry points. Instead, nodes can join as long as they can discover other members [37].

Some researchers strictly define the Ad hoc Grid as a grid environment without a fixed infrastructure: all its components are mobile [39, 40], as shown in Figure 2.5. This grid is referred to as the Mobile Ad hoc Grid. Details on Mobile Grids are presented in section 2.6.3.2.3. However, Ad hoc Grids focus more on the ad hoc nature of the grid rather than the mobility of its nodes.

The main challenge of an Ad hoc Grid is its dynamic topology, due to the rebooting of workstations, and the movement or replacement of computational nodes. More technical details concerning Ad hoc Grid challenges and implementations are available in [37-40].

Varying architectures have been proposed for Ad hoc Grids, for instance, [39] introduces a virtual backbone architecture that is dynamically constructed using nodes with high resource capacity. Other sources [37, 38] suggest Peer-to-Peer (P2P) architectures where computing resources are available on demand equally to every peer. Existing Ad hoc Grid projects include OurGrid [12] and myGrid [13].

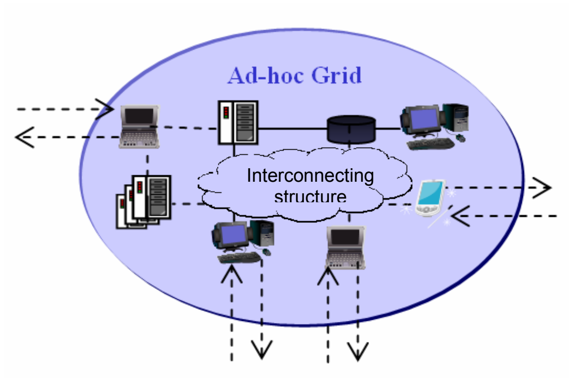


Figure 2.4: Ad hoc Grid

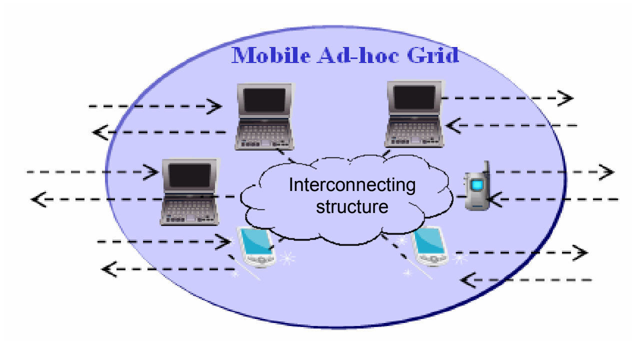


Figure 2.5: Mobile Ad hoc Grid

2.6.3.2.2 Wireless Grids

The Wireless Grid extends grid resources to wireless devices of varying sizes and capabilities such as sensors, mobile phones, laptops, special instruments and edge devices. They might be statically located, mobile or nomadic, shifting across institutional boundaries and connected to the grid via devices in close proximity [41].

In Wireless Grids, wireless devices can act as real grid nodes where part of data processing and storage is taking place, as shown in Figure 2.6. A special type of Wireless Grid is illustrated in Figure 2.7, in which all wireless devices are considered as pure access devices without processing or storage capabilities [42]; required resources are obtained from a wired resource-rich backbone grid.

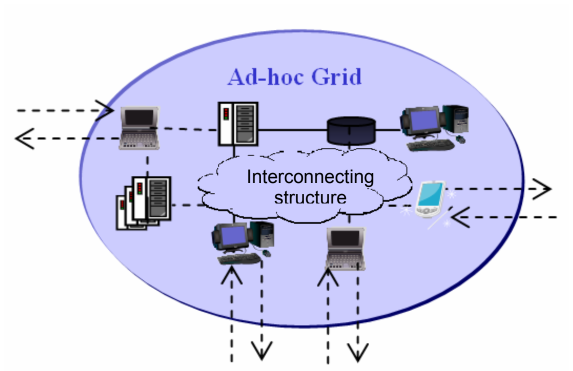


Figure 2.6: Wireless Grid

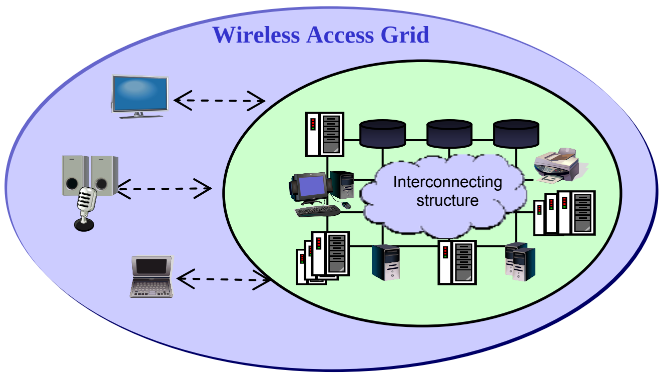


Figure 2.7: Wireless Access Grid

Many technical concerns arise when integrating wireless devices into a grid. These include high security risks, low bandwidth, power consumption and high latency. Therefore, several communities are exploring these new issues to ensure that future grid peers can be wireless devices [43]. Innovaticus [14] and FWGrid [15] are among the existing Wireless Grid projects.

2.6.3.2.3 Mobile Grids

Mobile Grids allow grid services to be accessible through mobile devices such as Personal Digital Assistants (PDAs) and smart phones, which are usually considered to be at best marginally relevant to grid computing. This is due to the fact that these devices are typically resource limited in terms of processing power, persistent storage, runtime heap, battery lifetime, screen size, connectivity and bandwidth. Thus, many researchers argue that mobile devices do not fit well into grid computing. In contrast, recent studies suggest a very different picture [44-53]. The millions of mobile devices sold annually should not be ignored and the raw processing power of some mobile devices is not insignificant given their mobility [44]. Furthermore, in emergency situations, such as natural disasters and battlefields, wireless mobile devices might be the only available communication and computation services. The most important argument is that, it is difficult to materialise the AmI vision without utilising such devices.

As in the case of wireless devices, there are already two approaches to integrate mobile devices into grid systems. In the first approach, the mobile nodes participate actively in the grid by providing computational or data services [45], as shown in Figure 2.8. This approach is what is usually referenced as "Mobile Grids". In the second approach, mobile devices serve as an interface to a stationary grid for sending requests and receiving results, as shown in Figure 2.9. Sometimes this approach is labelled "mobile access to grid infrastructure" [46] or simply Mobile Access Grids.

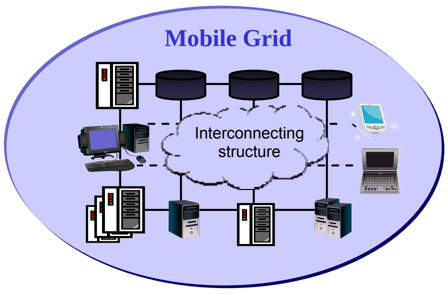


Figure 2.8: Mobile Grid

Recently, numerous efforts have been made towards establishing Mobile Grids. In [44-47] details concerning Mobile Grid requirements and challenges are presented. Various techniques have been proposed to implement the Mobile Grid vision from centralised [45] to P2P structure [48], from intelligent mobile agents [49] to mobile grid middleware [50] and many more. Some IST projects such as ISAM [51] and MADAM [52] have investigated issues related to mobility. However, the Akogrimo project [16] targeted Mobile Grids explicitly.

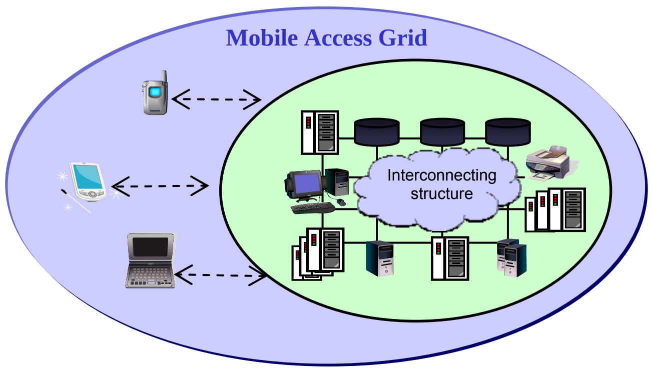


Figure 2.9: Mobile Access Grid

2.6.4 Grids Classified by Interactivity

Based on the mode of interaction, grids are classified into Batch Grids and Interactive Grids as shown in Figure 2.3.

2.6.4.1 Batch Grids

Batch Grids are traditional grid systems that do not support real time interactive sessions such as video gaming. Usually, they are implemented using Message Passing Interface (MPI) which consists of a set of libraries for parallel computing. Batch Grids employ queues in which the incoming parallel applications are stored before allocation by a batch scheduler to a set of processors for execution. Hence, the overall response time of an application is the sum of its queue waiting time and execution time.

2.6.4.2 Interactive Grids

Some potential application areas for NGG such as real-time embedded control systems and video gaming, require rapid response times and on-line interactivity which the classic request/response communication paradigm, in traditional grid systems, cannot accommodate [53]. Therefore Interactive Grids are emerging to extend the domain of grid application from traditional batch jobs to interactive sessions. Interactivity in grid environments can involve direct interaction between the grid and its end users; in this case the grid is labelled as a Direct Interactive Grid. However, this is only one form of possible interaction in grid environments. Another possible interaction is between a grid and its surroundings, referred to as Context-Aware Grids.

2.6.4.2.1 Direct Interactive Grids

In Direct Interactive Grids, end users interact with the grid through frequently submitting explicit requests to control or modify their running jobs, such as in CAD and video-gaming applications. This user interaction with the grid system can be implemented at two different levels: the Web portal level and grid middleware level. In the former, a Web-based grid portal is used to submit interactive jobs to a secure shell process, rather than directly to the grid middleware. ScGrid portal [54] falls into this category. In the latter, grid middleware is extended to support interactivity. Examples of this category include: CrossGrid [17], and edutain@grid [18].

2.6.4.2.2 Context-Aware Grids

In Context-Aware Grids, the interaction is between the grid and its environment. In such a grid, sensors are employed to interactively build the context through continuously gathering information from the surroundings. Controllers are utilised to analyse the information sent by sensors and instruct actuators to adapt grid behaviours accordingly. Many recent IST projects in networking, embedded and pervasive systems, such as SENSE [55] and MORE [56], have emphasised context awareness in their research agendas. However, CONTEXT [19] has a specific focus on grid environments.

2.6.5 Grids Classified by User-Centricity

As shown in Figure 2.3, in terms of user-centricity, grid systems are classified into User-Centric Grids and Organisational Grids.

2.6.5.1 Organisational Grids

Organisational Grids represent most traditional grid systems which are designed with professional expert users from research and enterprise domains in mind. They have highly sophisticated Web portals which hinders utilising them by inexperienced users.

2.6.5.2 User-Centric Grids

User-Centric Grids provide user-friendly grid environments for people in different domains. In grid computing, user-centricity could begin with the display of the end user's name on a Web portal, and might end with the personalisation of all information, resources and networks underpinning grids. Two categories of User-Centric Grids can be identified: Customisable Grids and Personal grids. Personal Grids have already been presented in section 2.6.2.5.

Customisable Grids are designed with highly personalisable Web portals to provide user-friendly access points to grid resources for people in different domains. However, research to support Customisable Grids is in its infancy. In the myGrid project [13], scientists are allowed to establish multiple views which provide access to a user-defined subset of the registered services. These views can be specific to individual scientists or to further, more specialised, discovery services. In the Akogrimo project [16], profiles and special needs for all learners are kept and automatically loaded whenever they sign on, providing a customised user-friendly environment for each learner.

2.6.6 Grids Classified by Manageability

A grid is highly complex and dynamic in nature, making its management extremely challenging. A variety of technologies are available to support grid manageability at both hardware and software levels. At the software level, manageability can be achieved with a wide range of techniques from traditional log files, to recent technologies such as Java Management Extensions (JMX) [57] and knowledge technologies [58]. At the hardware level, this can be achieved with technologies from simple embedded sensors [59] to stand-alone intelligent robots. Additionally, manageability might be supported by changing the underlying grid architecture, for example, from centralised client/server to P2P [60] structures.

Grid management is concerned mainly with service and resource management. Therefore grid resource management systems are considered as the heart of any grid environment. In [27] a comprehensive taxonomy which classifies resource management systems based on ten criteria is presented. In this chapter we are concerned with classifying grid systems in general. We based our classification in regard to manageability on the management scheme of the scheduler which is the core of resource management systems.

Figure 2.3 shows that, according to the scheduler management strategy, grids are classified into: Centralised Grids, P2P Grids, Manageable Grids and Hybrid Grids.

2.6.6.1 Centralised Grids

Typically, Grids are centralised systems with one entity making decisions for the whole system. Traditional approaches to grid management require centralised servers, extensive knowledge of the underlying systems and a large group of experienced staff. Although this scheme has the advantages of simple deployment and ease of control, it suffers severely from lack of scalability and fault tolerance.

2.6.6.2 P2P Grids

In contrast to Centralised Grids, P2P Grids remove any form of centralised authority. All grid nodes are under distinct or even unrelated control; they can decide to join or leave at any time. Therefore, P2P Grids are highly dynamic in nature requiring special algorithms and strategies. Within P2P Grids, each peer acts as an autonomous entity but depends on other peers for resources, information, and forwarding requests. The main goal of a P2P Grid is to ensure scalability and reliability. Many P2P Grids are concerned with content and file-sharing focusing on creating efficient strategies to locate particular files, providing reliable transfers of such files and managing high load caused by demand for highly popular files [29].

2.6.6.3 Manageable Grids

In this context, a Manageable Grid is defined as a sophisticated grid that automatically manages, adapts, monitors, diagnoses and fixes itself. Manageable Grids offer a simplified installation and greatly reduce configuration and administration which in turn reduce management costs and dramatically enhance scalability. Existing research in this area is classified into Autonomic

Grids, Knowledge Grids and Organic Grids, as shown in Figure 2.3 and Table 2.1.

2.6.6.3.1 Autonomic Grids

Autonomic computing [61], initiated by IBM in 2001, is named after the human body's autonomic nervous system. An autonomic computing system controls the functioning of computer systems without users' intervention; likewise the autonomic nervous system regulates body systems without any external help. The main goal of autonomic computing is to reduce the complexity of the management of large computing systems, such as the grid [62].

An Autonomic Grid is a grid that is able to configure, re-configure, protect and heal itself under varying and unpredictable conditions. It can optimise its work to maximise resource utilisation. Applications, challenges and various methods that have been proposed to work towards Autonomic Grids are presented in [23]. Examples of Autonomic Grid projects include The IBM OptimalGrid [22] and AutoMAGI [23].

2.6.6.3.2 Knowledge Grids

A Knowledge Grid is an extension to the current grid in which data, resources and services are given well-defined meanings that are understandable at both machine and human levels using semantic metadata and ontology. The aim is to move the grid from an infrastructure for computation and data management to a pervasive, knowledge management infrastructure. Examples of Knowledge Grid projects include OntoGrid [24] InteliGrid [25] and K-Wf Grid [63]. Several communities are working to realise knowledge Grids including the Semantic Grid Group [64] from the OGF [65]. Reviews of the current status and future vision of knowledge Grids, including applications, challenges and critical issues, are detailed in [66, 67].

2.6.6.3.3 Organic Grids

Traditionally, 'organic' means forming an integral element of a whole; having systematic coordination of parts; having the characteristics of an organism and developing in the manner of a living plant or animal [68]. In grid computing, the

Organic Grid comes to refer to a new design for Grid systems that relies on a decentralised P2P approach, distributed scheduling scheme and mobile agents. The basic idea is taken from the manner in which complex patterns can emerge from the interplay of many agents. A framework for a Desktop Grid based on an ant colony is presented and evaluated in [26].

2.6.6.4 Hybrid Grids

Hybrid Grids use different combinations of management schemes. For instance, a grid environment may implement a distributed P2P management scheme at the cluster level while the management structure at the higher grid level is centralised.

2.7 Other Related Paradigms

Originally, the term grid computing started as a metaphor for making computer power as easy to access as an electric grid [1]. This has the advantage of a low, or no initial, cost to acquire hardware; instead, computational resources are essentially rented on demand. Indeed, this idea of offering computing resources, such as computation power and storage spaces, as a metered service similar to public utilities such as electricity, water and telephone network, is not unique to grid computing. Rather, it is the driving vision of other distributed system paradigms, namely: utility (on-demand) computing, cloud computing and everything as a service (EaaS/XaaS/aaS). These terms are often confused with grid computing or used as synonyms for it.

2.7.1 Utility Computing

The main difference between grid computing and utility computing (On-Demand) resides in the definition of the two terms [69]. Grid computing is a distributed system infrastructure (hardware and software) for enabling remote resource sharing and utilisation to provide massive computing capabilities as a set of services. Utility computing is a service provisioning model where computing resources are offered as utility services in terms of availability, ease of access, on demand usage, and billing schemes. In this sense, grid computing can serve as the enabling technologies and environments for utility computing.

In turn, the utility model increases the efficiency of grid resource utilisation by acquiring them only when a demand arises. Utility Grids (also known as Service Grids), as presented in section 2.6.3.1, is an example of this strong relationship between grid and utility computing.

Several advantages are offered by the utility computing service provisioning model for both service consumers and providers. Service providers do not set up or configure hardware and software components for a single application or user; instead virtual resources are dynamically allocated and reallocated to a large user community based on their needs. This increases the resources utilisation and decreases the operational cost. From a user's perspective, utility computing excuses them from heavily investing in building, operating and maintaining a computing infrastructure. Additionally, users do not need to concern themselves with resource management and utilisation [70].

2.7.2 Everything as a Service

Nowadays, Service–Oriented Architecture (SOA) has become the main architectural model of many IT initiatives including grid, cloud and everything as a service (EaaS/XaaS/aaS) computing. The SOA does not specify implementation technologies or platforms, although it is usually coupled with Web services, but rather it is an architectural approach for constructing software systems from a set of smaller building blocks called services. The goal is to have software systems which are implementation agnostic with loose coupling and interoperability among different software components [71].

Application Service Providers (ASP) have adopted the basic idea of service orientation, by hosting loosely coupled software components, i.e. services, which can be accessed on-demand, and coined the term Software as a Service (SaaS) to refer to it. Primarily, SaaS is employed to obtain rights to use software on demand which alleviates the customer's liability for licensing all devices with all applications. SaaS has been also applied successfully in other application areas such as e-mail, customer relationship management (CRM), and web content management. SaaS is the oldest model of the XaaS. Other

XaaS models include Communication, Infrastructure and Platform as a service among others.

Communication as a Service (CaaS) is a generic term for several different but related services. Under the broad CaaS umbrella, comes Voice over IP (VoIP also sometimes referred to as Voice as a Service (VaaS)), remote automated call distribution (ACD) and hosted Private Branch Exchange (PBX), among others.

Infrastructure as a Service (IaaS), initially known as Hardware as a service (HaaS), is a new idea in XaaS. It aims to replace critical data centre resources such as physical servers and storage spaces with scalable and highly-available resources in the Internet. These resources are allocated dynamically based on users' demand [72]. The most known example of IaaS is Amazon's EC2 (Elastic Compute Cloud) [73] and GoGrid Cloud Hosting Services [74].

Platform as a Service (PaaS), also known as cloudware, is the newest kind of services within the XaaS collection. Its main aim is to allow building and delivering entire web applications and services through the Internet without downloading or installing any developer's platform. Known PaaS examples include Google's AppEngine [75] and Salesforce's force.com [76].

The common thread amongst all these XaaS services is the outsourcing and on-demand nature of their offerings. XaaS is a service deployment and provisioning model that can be viewed as a class of, or a more recent term for, utility computing. Grid computing technologies and platforms can be utilised to implement and provide XaaS services and platforms. XaaS services constitute the majority of cloud computing elements. In turn, cloud computing can manage/provide XaaS services.

2.7.3 Cloud Computing

The term cloud computing originates from the fluffy cartoonish cloud that usually appears at the middle of network diagrams. Recently, the cloud computing term has been adopted to refer to Internet style computing. Cloud computing is a general concept that incorporates the SOA, XaaS, outsourcing, and other recent known technology models where the common theme is the reliance on the Internet for satisfying the computing demands of the users [77].

There is still no clear or agreed definition of cloud computing, despite the fact that it has attracted vast attention. In [78] cloud computing is defined as "a paradigm in which information is permanently stored in servers on the Internet and cached temporarily on clients that include desktops, entertainment centres, table computers, notebooks, wall computers, handhelds, sensors, monitors, etc."

This ambiguity and indetermination of the cloud computing definition and edges, increases the confusion between cloud, grid, utility and XaaS Computing. However, while utility and XaaS computing are more about service provisioning models, cloud computing, in accord with the grid, is about platforms and technologies for offering computing resources. Both, grid computing and cloud computing, need to tackle the same problems such as managing a large pool of computing facilities and defining methods for service provisioning and discovery. Both utilise the same techniques such as resource virtualisation. Actually, grid computing is often associated with the delivery of cloud computing systems and cloud computing can provide physical and virtual servers on which the grid application can run. Indeed, many of today's cloud computing deployments are powered by grids, composed mainly of XaaS components and are built like utilities.

2.8 Conclusion

In this chapter a classification of emergent grids is presented. Representative projects were reviewed and classified. Such a classification assists in detailed comparisons between emerging grids. It helps in understanding current research in grid computing and anticipating future trends. The review also assists in identifying the key implementation approaches and issues related to each emerging grid.

However, most emerging grids are still in their infancy stage of development. This study indicates the necessity for more research in this domain, so as to establish a solid background and enable the implementation of these promising environments.

The proposed classification for emerging grids has been extended in the form of a comprehensive taxonomy to accommodate both traditional and emerging grids. Such taxonomy has the potential to allow a comparison of past, current and future work in grid computing based on one scheme. The intention has been to provide a common set of terminologies and classification scheme in the rapidly evolving area of grid computing.

Some emerging grids share features with PM-Grids, namely: Personal Grids, Mobile grids and Organic Grids. However, Personal Grids target individual users but do not address the mobility issue. Mobile Grids address the mobility issue but do not consider individuals among their users. Organic Grids focus on self-management problems through ideas from social insects but consider neither personal users nor mobile devices.

Utility computing, XaaS and cloud computing have recently emerged with the same vision as grid computing. While utility computing and XaaS are service provisioning models, both grid computing and cloud computing offer architectures and technologies for distributed computing. Thus it is still difficult to define clear boundaries between the two, especially when it comes to emerging grids. At the present stage, it seems as though cloud is a new commercial name for a grid. The commercial reality is that new names sometimes enable development as they initiate renewed discussion and attract funding.

2.9 References

[1] I. Foster and C. Kesselman, Eds., *The Grid2: Blueprint for a Future Computing Infrastructure*. San Francisco: Morgan Kaufmann, 2003.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organisation," *Int. J. Supercomput. Applicat.*, vol. 15, no. 3, pp. 200-222, 2001.

[3] J. Joseph and C. Fellenstein, *Grid Computing*, IBM Press, 2004.

[4] B. Rajkumar, "Grid computing: Making the global cyberinfrastructure for eScience a reality," *CSI Commun.,* vol. 29, *2005.*

[5] Sun Microsystems [online]. Available: http://www.sun.com/, [accessed Feb. 2, 2010].

[6] Sun Microsystems, (2002, May). Sun cluster grid architecture [online]. Available: http://www.sun.com/software/grid/SunClusterGridArchitecture.pdf, [accessed Feb. 2, 2010].

[7] I. Foster and C. Kesselman, Eds. *The Grid: Blueprint for a Future Computing Infrastructure.* San Francisco: Morgan Kaufmann, 1999.

[8] K.G. Jeffery, "Next generation grids for environmental science," *Environmental Modelling & Softw.,* vol. 22, no. 3, pp. 281–287, 2007.

[9] Expert Group, "Next generation grids: European grid research 2005-2010," Expert Group Rep., Jun. 2003 [online]. Available: ftp://ftp.cordis.lu/pub/ist/docs/ ngg_eg_final.pdf, [accessed Feb. 2, 2010].

[10] Expert Group, "Next generation grids2: Requirements and options for European grids research 2005-2010 and beyond," Expert Group Rep., Jul. 2004 [online]. Available: http://www.semanticgrid.org/docs/ngg2_eg_final.pdf, [accessed Feb. 2, 2010].

[11] Expert Group Final, "Future for European grids: Grids and service oriented knowledge utility," Expert Group Final Rep., Jan. 2006 [online]. Available: ftp://ftp.cordis.europa.eu/pub/ist/docs/grids/ngg3_eg_final.pdf, [accessed Feb. 2, 2010].

[12] OurGrid [online]. Available: http://www.ourgrid.org, [accessed Feb. 2, 2010].

[13] myGrid [online]. Available: http://www.mygrid.org.uk/, [accessed Feb. 2, 2010].

[14] Innovaticus [online]. Available: http://wglab.net/research, [accessed Feb. 2, 2010].

[15] FWGrid [online]. Available: http://fwgrid.ucsd.edu/, [accessed Feb. 2, 2010].

[16] Akogrimo [online]. Available: http://www.mobilegrids.org/, [accessed

Feb. 2, 2010].

[17] CrossGrid [online]. Available: http://www.crossgrid.org/, [accessed Feb. 2, 2010].

[18] edutain@grid project [online]. Available: http://www.edutaingrid.eu/, [accessed Feb. 2, 2010].

[19] CONTEXT [online]. Available: http://context.upc.es/ [accessed Feb. 2, 2010].

[20] J. Han and D. Park, "A lightweight personal grid using a supernode network", in *Proc. 3rd Int. Conf. P2P 2003,* pp. 168-175.

[21] Z. Xu, L. Xiao and X. Liu, "Personal Grid," *in Proc. NPC*, 2007, pp. 536-540.

[22] T. J. Lehman and J. H. Kaufman, "OptimalGrid: Middleware for automatic deployment of distributed FEM problems on an internet-based computing grid," in *Proc. IEEE Int. Conf. Cluster Comput.,* 2003, pp. 164-171.

[23] A. Sajjad, H. Jameel, U. Kalim, S. Han, Y. Lee and S. Lee, "AutoMAGI - an autonomic middleware for enabling mobile access to grid infrastructure," in *Proc. ICAS-ICNS 2005*, pp. 73-79.

[24] OntoGrid [online]. Available: http://www.ontogrid.net/ontogrid/index.jsp, [accessed Feb. 2, 2010].

[25] InteliGrid [online]. Available: http://www.inteligrid.com/, [accessed Feb. 2, 2010].

[26] A. J. Chakravarti, G. Baumgartner and M. Lauria, "The organic grid: Self-organizing computation on a peer-to-peer network," *IEEE Trans. Syst. Man Cybern. A, Syst. Humans,* vol. 35, pp. 373-384, 2005.

[27] K. Krauter, R. Buyya and M. Maheswaran, "A Taxonomy and survey of grid resource management systems for distributed computing," *Softw. Prac. Exper.*, vol. 32, no. 2 , pp. 135-164, 2002.

[28] A. J. Wells. *Grid Application Systems Design.* FL: CRC Press, 2007.

[29] S. Venugopal, R. Buyya and K. Ramamohanarao, "A taxonomy of Data Grids for distributed data sharing, management, and processing," *ACM Comput. Survey,* vol. 38, pp. 3, 2006.

[30] 36 national Grids initiatives in Europe support the EGI concept, EnterTheGrid-PrimeurMagazine, (2007, Sep.) [online]. Available: http://enterthegrid.com/primeur/07/articles/monthly/AE-PR-10-07-48.html, [accessed Feb. 2, 2010].

[31] G. Fox and D. Walker, "E-science gap analysis," UK e-Science Tech. Rep. Series, Jun. 2003.

[32] Enterprise Grid Alliance, "Enterprise grid alliance reference model v1.0," Apr., 2005 [online]. Available: http://www.ogf.org/UnderstandingGrids/documents/EGA_reference_model.pdf, [accessed Feb. 2, 2010].

[33] I. Baird, (2003, Jun.). Grids in Practice: A platform perspective, *MIDDLEWARESpectra*, [online]. Available: http://www.middlewarespectra.com/grid, [accessed Feb. 2, 2010].

[34] W. Li, Z. Xu, B. Li, Y. Gong, "The Vega Personal Grid: A lightweight grid architecture," *in Proc. IASTED*, 2002, pp. 6-11.

[35] B. Li, W. Li and Z. Xu, "Personal Grid running at the edge of internet," *in Proc. 2nd GCC*, 2003, pp. 762-769.

[36] Globus Alliance [online], (2007, Oct.). Available: http://www.globus.org/, [accessed Feb. 2, 2010].

[37] K. Amin, G. Laszewski, A. R. Mikler, "Toward an architecture for ad hoc grids," in *Proc. IEEE 12th Int. Conf. ADCOM 2004*, Ahmedabad Gujarat, India, 2004. Available: http://www.mcs.anl. gov/_gregor/papers/vonLaszewski-adhoc-adcom2004.pdf, [accessed Feb. 2, 2010].

[38] T. Friese, M. Smith and B. Freisleben, "Hot service deployment in an ad hoc grid environment," in *Proc. ICSOC,* 2004, pp. 75-83.

[39] D. C. Marinescu, G. M. Marinescu, J. Yongchang, L. Boloni and H. J. Siegel, "Ad hoc grids: Communication and computing in a power constrained environment," in *Proc. IEEE Int. Performance, Comput. Commun. Conf.,* 2003, pp. 113-122.

[40] S. Shivle, H. J. Siegel, A. A. MacIejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment," *J. Parallel Distrib. Comput.,* vol. 66, pp. 600-611, 2006.

[41] L. W. McKnight, J. Howison and S. Bradner, "Wireless grids: Distributed resource sharing by mobile, nomadic, and fixed devices," *IEEE Internet Comput.,* vol. 8, pp. 24-31, 2004.

[42] S. H. Srinivasan, "Pervasive wireless grid architecture," in *Proc. 2nd Annual Conf. Wireless on-Demand Network Syst. Services,* 2005, pp. 83-88.

[43] J. Hwang and P. Aravamudham, "Middleware services for P2P computing in wireless grid networks," *IEEE Internet Comput.,* vol. 8, pp. 40-46, 2004.

[44] T. Phan, L. Huang and C. Dulan, "Challenge: Integrating mobile wireless devices into the computational grid," in *Proc. 8th Annu. Int. Conf. Comput. Networking,* 2002, pp. 271-278.

[45] K. Ohta, T. Yoshikawa, T. Nakagawa and H. Inamura, "Design and implementation of mobile grid middleware for handsets," in *Proc. 11th Int. Conf. Parallel Distrib. Syst.,* 2005, pp. 679-683.

[46] A. Sajjad, H. Jameel, U. Kalim, Y. Lee and S. Lee, "A component-based architecture for an autonomic middleware enabling mobile access to grid infrastructure," in *Proc. EUC 2005*, pp. 1225-1234.

[47] Bhagyavati and S. Kurkovsky, “Emerging issues in wireless computational grids for mobile devices,” in *Proc. 8th World Multiconf. SCI-2004*. Available: http://www.cs.ccsu.edu/~stan/research/Grid/PubsSCI2004.pdf, [accessed Feb. 2, 2010].

[48] L. Lima, A. Gomes, A. Ziviani, M. Endler, L. Soares and B. Schulze, “Peer-to-peer resource discovery in mobile Grids,” in *Proc. 3rd int. workshop MGC '05*, 2005, pp. 1-6.

[49] S. Kurkovsky, Bhagyavati and A. Ray, “Modelling a grid-based problem-solving environment for mobile devices,” *J. Digital Inform. Manage.,* vol. 2, pp. 109-114, 2004.

[50] D. C. Chu and M. Humphrey, “Mobile OGSI.NET: Grid computing on mobile devices”, in *Proc. 5th IEEE/ACM Int. Workshop Grid Computi.,* 2004, pp. 182-191.

[51] ISAM [online]. Available: http://www.inf.ufrgs.br/~isam/English/, [accessed Feb. 2, 2010].

[52] MADAM [online]. Available: http://www.ist-madam.org/, [accessed Feb. 2, 2010].

[53] V. Talwar, S. Basu and R. Kumar, “Architecture and environment for enabling interactive grids,” *J. Grid Comput.,* vol. 1, pp. 231-250, 2003.

[54] H. Xiao, H. Wu, X. Chi, S. Deng and H. Zhang, “An implementation of interactive jobs submission for grid computing portals”, in *Proc. AusGrid2005,* pp. 67-68.

[55] SENSE [online]. Available: http://www.sense-ist.org/, [accessed Feb. 2, 2010].

[56] MORE [online]. Available: http://www.ist-more.org/, [accessed Feb. 2, 2010].

[57] GigaSpaces Project [online]. Available: http://www.psdesign.co.il/gigaspaces/, [accessed Feb. 2, 2010].

[58] J. Nichols, H. Demirkan and M. Goul, “Autonomic workflow execution in the grid,” *IEEE Trans. Syst. Man and Cybern. C, Appl. Rev.,* vol. 36, pp. 353-364, 2006.

[59] IBM, “OGSI-Based System Management: Manageability Services for Linux,” white paper, IBM, Aug. 2003 [online]. Available: http://whitepapers.silicon.com0,39024759,60109062p,00.htm, [accessed Feb. 2, 2010].

[60] A. J. Chakravarti, G. Baumgartner and M. Lauria, “Self-organizing scheduling on the organic grid,” *Int. J. High Performance Comput. App.,* vol. 20, pp. 115-130, 2006.

[61] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *IEEE Comput.,* vol. 36, pp. 41-50, 2003.

[62] M. Li and M. Baker, *The Grid: Core Technologies.* Wiley, 2005.

[63] K-Wf Grid [online]. Available: http://www.kwfgrid.eu/, [accessed Feb. 2, 2010].

[64] Semantic Grid Community Portal [online]. Available: http://www.ogf.org/gf/group_info/view.php?group=sem-rg, [accessed Feb. 2, 2010].

[65] Open Grid Forum [online]. Available: http://www.ogf.org/, [accessed Feb. 2, 2010].

[66] D. Roure, N.R. Jennings and N.R. Shadbolt, "The Semantic Grid: Past, Present, and Future," in *Proc. IEEE*, vol. 93, no. 3, pp. 669-681, 2004.

[67] M. Geldof, "The semantic grid: will semantic Web and grid go hand in hand," The Grid technologies unit of the European Commission, Jun. 2004 [online]. Available: http://www.semanticgrid.org/documents/Semantic%20Grid%20report%20public.pdf, [accessed Feb. 2, 2010].

[68] E. Tofslie, "Organic Grid Design," BFA thesis, College of Letters, Arts, and Social Sciences, Univ. of Idaho, 2002. Available: http://www.tofslie.com/organicgrid.pdf, [accessed Feb. 2, 2010].

[69] Clabby Analytics, "The grid report," May 2004 [online]. Available: http://www.ibm.com/grid/pdf/Clabby_Grid_Report_2004_Edition.pdf, [accessed Feb. 2, 2010].

[70] C. S. Yeo, M. Dias de Assuncao, J. Yu, A. Sulistio, S. Venugopal, M. Placek, and R. Buyya, "Utility computing and global grids," in *The Handbook of Computer Networks*, H. Bidgoli, Ed., New York: John Wiley & Sons, 2007.

[71] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, 2005.

[72] LINUX Magazine [online]. Available: http://www.linux-mag.com/id/7197, [accessed Feb. 2, 2010].

[73] Amazon Elastic Compute Cloud (Amazon EC2) [online]. Available: http://aws.amazon.com/ec2/, [accessed Feb. 2, 2010].

[74] GoGrid Cloud Hosting Services [online]. Available: http://www.gogrid.com/, [accessed Feb. 2, 2010].

[75] Google App Engine [online]. Available: http://code.google.com/appengine/, [accessed Feb. 2, 2010].

[76] Salesforce's force.com [online]. Available: http://www.salesforce.com/uk, [accessed Feb. 2, 2010].

[77] V. Šipková and M. Dobrucký: "Towards an Advanced Distributed Computing," *in Proc. GCCP'2008,* pp. 128-131.

[78] C. Hewitt, "ORGs for scalable, robust, privacy-friendly client cloud computing", *IEEE Internet Comput. online*, vol. 12, no. 5, pp. 96-99, Sep/Oct, 2008.

Chapter 3
PM-Grid: A Personal Mobile Grid

3.1 Introduction

Notwithstanding the escalating popularity of grid computing in both research and enterprise domains, personal users, i.e. individuals outside these domains, are still not supported. On the other hand, new applications and complicated problems are increasingly emerging in everyday life where no computational tools are available but mobile devices. Creating a means to bridge the gap between computational grids and personal users with resource limited mobile devices is the core of PM-Grids.

The major contribution of this chapter is to introduce PM-Grids as a new paradigm in grid computing for individuals constrained by devices of limited resources. Section 3.2 lays the background of PM-Grids while section 3.3 defines them. In section 3.4 and section 3.5 the motivating applications and main debating issues related to integrating personal mobile devices in grid environments are addressed respectively. Architectural designs of PM-Grids to exploit resources available within PNs are presented in section 3.6. Section 3.7 compares PM-Grids with related works in the area.

3.2 From Mainframes to PM-Grids

Electronic digital computers emerged as massive building-sized machines, as shown in Figure 3.1. They were known as “mainframes”. They started at scientific research centres then were adopted by the business domain. At that point of time, the idea of a mobile personal computer, something small and light enough for individuals to pick and carry around, was not acceptable even by academic researchers or leading computer companies. They did not consider a personal mobile computer allowing computational capabilities for individuals while travelling to be an idea worth pursuing. Simply, they could not see any practical purpose for such a device [1].

Figure 3.1: ENIAC, the Second Electronic Digital Computer, 1943 [2]

Nowadays, the personal mobile computer market has become one of the largest markets in the world. It is rapidly evolving with progressive reduction in cost, weight and size and continuous improvement in performance. This has enabled many people to move around with a basic set of electronic gadgets which usually includes a mobile phone, PDA and laptop. These devices, which belong to the same user and are usually within ten metres of her/him, can be connected together with available wired connectivity, such as USB and FireWire, or wireless technologies, such as IrDA, Bluetooth, UWB, ZigBee and other technologies included in the IEEE 802.15 family of standards [3]. This network with the user at its inner core, which is known as a Personal Area Network (PAN) [4], is illustrated in Figure 3.2.

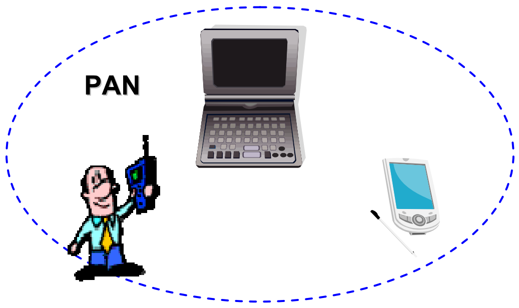


Figure 3.2: Personal Area Network (PAN)

Besides this basic set of electronic devices within the PAN, one might have other devices in different locations, for instance in the home, office and car.

These devices, which belong to the same user, can be connected together regardless of their geographical locations to form a Personal Network (PN) [5, 6]. Thus, one can gain access to his/her electronic devices, any time anywhere, and can share resources among them, as shown in Figure 3.3.

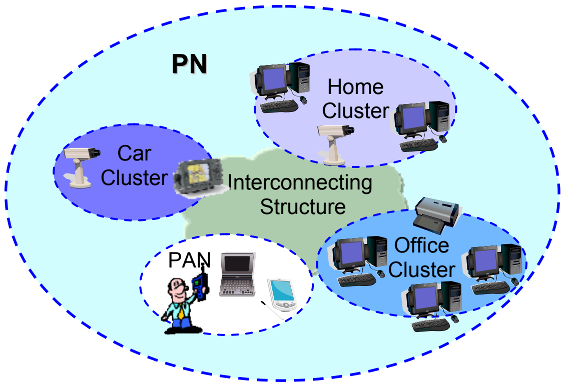


Figure 3.3: Personal Network (PN)

Nonetheless, PANs and PNs are most commonly used for applications involving data and peripheral sharing. This is due to the resources allowed for sharing in PNs, PANs and all today’s conventional networks being limited to data, peripherals and secondary storage. The most important resources, namely, processors cycles and runtime memories, are still not available for sharing across these networks.

Hence, an important question arises here: Why not further enable these networks to seamlessly share other resources such as processing cycles, storage capacity and functionality in the form of services available across computational grids? As PNs can already share data, peripherals and secondary storage among their devices, the next logical step is to superimpose grid functionality over them to allow the sharing of processors cycles and memories. Thus the net result is a huge virtual computer which can be accessed at anytime from anywhere. That is to say, a Personal Mobile Grid, as shown in Figure 3.4.

Grid computing systems have started exactly the same way as digital computers; emerging as massive computing facilities in scientific research centres before being adopted by large commercial enterprises. Nevertheless, many people even from the grid computing community may not be able to understand why an individual might need to have their own grid with great computational capabilities while on the move. This should not hinder thinking about such a potential system, a PM-Grid. In section 3.4 some potential motivating applications for PM-Grids are presented.

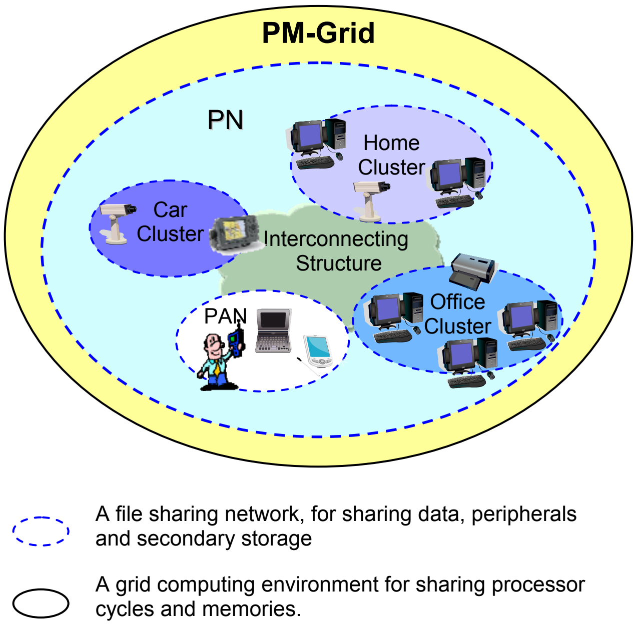

PM-Grid
PN
Home Cluster
Car Cluster
Interconnecting Structure
Office Cluster
PAN
A file sharing network, for sharing data, peripherals and secondary storage
A grid computing environment for sharing processor cycles and memories.

Figure 3.4: Personal Mobile Grid (PM-Grid)

3.3 What is a PM-Grid?

A PM-Grid is a grid environment which can be owned and utilised by an individual user. It is constructed over his/her devices and might be extended to other devices which s/he trusts. PM-Grids aim to enable the mobility of both, users requesting access to grid resources and resources that are part of a grid. Hence, the distinguishing characteristic of a PM-Grid is that it is primarily constructed, owned and utilised by an individual (or a group of individuals with a mutual trust relationship). This is in contrast to traditional grids which are constructed, owned and utilised by organisations and other large entities. In other words, where traditional grids are concerned with a large user population, the PM-Grid is only concerned with a single user. Also the type of application is different; where traditional grids are chiefly concerned with massive complex world-wide computations, PM-Grid applications are considerably smaller in size, scope and complexity. Additionally, where traditional grids need a well-established stationary infrastructure to operate, PM-Grids can be fully accommodated in mobile devices connected via a PN.

3.4 Motivating Applications

As indicated in Chapter 1, people are increasingly keen to frequently replace or upgrade their personal computers to gain more processing power and memory. Sometimes they need to run complex computational jobs which their PC or laptop cannot accommodate, or while they are travelling away from home. People are becoming frustrated with the need to move data between their different electronic devices, such as for instance, a person having several address books scattered among his/her devices. Indeed, there is a need to allow a user to harness all processing powers, memory storages and data files distributed across his/her computing and communication devices. A PM-Grid has the potential to satisfy this need.

Other contexts might also be considered where a small business needs to run an intensive forecasting simulation to make critical financial decisions, or a large charity group co-ordinating a large multimedia database. Hence, PM-Grids can be utilised equally by individuals as well as small groups. In a nut shell, PM-Grids are intended to serve as general purpose computing environments available for individuals or a VO of a very limited scope, any time anywhere, exactly as personal mobile computers, with the additional services and huge computing capability available due to the aggregated networked resources.

As stated earlier, PM-Grids allow the mobility of both users requesting access to grid resources, and resources that are themselves part of a grid. This opens the doors to have the grid processing power in more widespread geographical locations and social settings, such as emergency communications in fire fighting and natural disasters, as well as many of the newly emerged mobile applications in e-learning, e-healthcare, e-wallet, and m-gaming, among others. In [7] a comprehensive illustration of application scenarios for Mobile Grids is presented, in which PM-Grids might be even more efficiently applied in terms of security and responsiveness since all PM-Grid resources are dedicated to a single user. This section sheds more light on two potential applications of PM-Grids: Personal Mobile Medical Record and (PM-MR) and Personal Mobile Learning (PM-learning).

3.4.1 Personal Mobile Medical Record (PM-MR)

In [8], a discussion of general challenges in implementing grid functionalities in a mobile environment, and the specific issues arise from a realistic e-healthcare emergency scenario, was presented. The PM-Grid infrastructure can play an important role in serving both patients and physicians in/outside hospitals. Here, the PM-MR is presented as a motivating example of how PM-Grids can be exploited in healthcare contexts.

At present, patient medical records (MRs) may be scattered in different locations; without access to them all at the same time. A patient might have MRs in multiple hospitals and clinics around the world. Sharing of patients' MRs among hospitals is important in many situations. For instance, medical history, current medications, allergies, etc. are always useful for doctors prescribing medications.

Some work has already been done in developing forms of health smart cards [9, 10] and Web-based MRs [11, 12]. The problem with a health smart card is that it adds to the number of cards the individual needs to carry around and might be lost or missed at any time. It needs special hardware to read and is considerably limited in terms of capacity. The problem with Web-based medical record is that they are not integrated, which means that a patient might have multiple electronic medical records; one in each hospital providing health services to the patient. Also, Web-based MRs suffer from the accessibility problem; they require the availability of a computer system with Internet connection in order to be viewed or manipulated. In [13] a software technology is proposed and is under development to allow mobile phone and PDA users to download their MRs and display animated 3D scans. However, the problem of the multiplicity of MRs is still exists.

Indeed, having access to all a patient's MRs as a single virtual MR anytime anywhere is consistent with efficient healthcare. So, a unified virtual copy of all

MRs that belongs to the same patient is stored in a well secured location in his PN and synchronised automatically with his/her physical MRs, as shown in Figure 3.5.

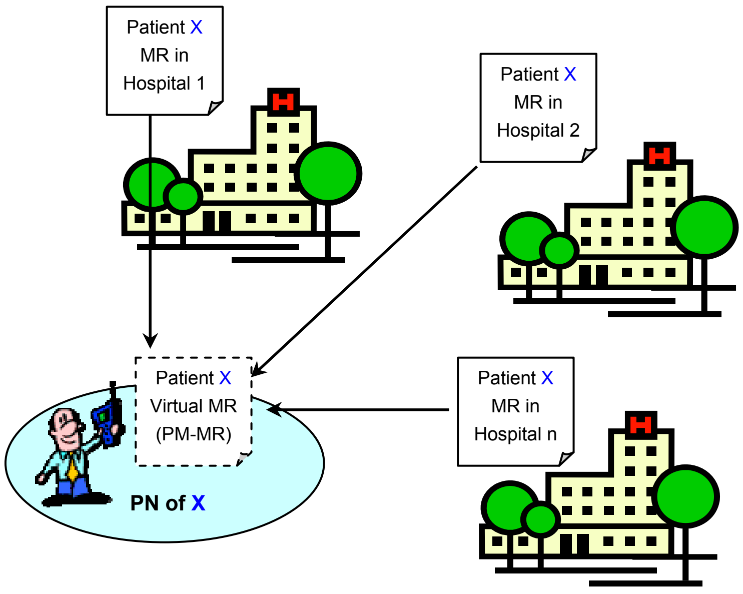


Figure 3.5: Personal Mobile Medical Record (PM-MR)

The patient can use his/her PDA or smart mobile phone to access their MR with a certain privilege, just as the doctors may access using a different level of privilege. The PM-MR can also remind the patient of times for medications, or medical appointments. It can be updated as new services are performed and new medications are prescribed, and much more.

The PM-MR efficiency can be boosted through including a wearable computing device, a small body-worn computer with sensors, in the core PAN so the PM-MR can be intelligent enough to instantly monitor and analyse a patient's data and alert her/him of any potential health hazard. It can contact people on call in any emergency situation, guide them to the patient's location, then help to analyse and visualise any necessary medical data and images. It can make appointments, or effect cancellations on behalf of the patient, having access to his/her e-diary.

Knowledge technologies, such as metadata and ontology, are very important for patient's data annotation and would play a focal role in PM-MRs. However, it is extremely important to note that such an application involves enormous critical security and ethical issues which need to be resolved.

3.4.2 Personal Mobile Learning (PM-Learning)

There are a number of scenarios where PM-Grids have the potential to enhance the e-learning experience, such as:

- Providing mobile access to existing learning objects, such as course content, exercises and exams among others.
- Providing mobile access to computing–intensive simulations, for engineers for instance.
- Allowing heavy multimedia content to be received by small devices
- Enhancing collaboration by gathering interactive services such as SMS, MMS, emails, and chat, among others.

Thus electronic learning and training would be available not only at well-equipped institutions, but also at remote locations, on the move or in emergency situations. This is valuable for students as well as company employees, accessing on-line training or instruction at remote locations, or tourists eager to learn more about regions to be visited and explored.

3.5 Grid Computing and Personal Mobile Devices

In Chapter 2, Section 2.6.3.2.3, Mobile Grids were briefly reviewed pointing to some of the challenges integrating personal mobile devices in grid environments. This section elaborates and discusses issues related to integrating such devices in grid environments.

Utilising mobile devices in grid environments has raised several debating issues between grid computing practitioners:

- Can mobile devices be utilised in grids?
- What roles can a mobile device play in this case?
- How can mobile devices be integrated in current grids?

In the following sections these issues are investigated further.

3.5.1 Can Mobile Devices be Utilised?

Ideally, to deploy a grid, powerful computational resources are combined to form a large-scale distributed system in which all resources, including processor cycles and memories, are shared. As a consequence, many researchers consider mobile devices as at best only marginally relevant to grid computing. This is due to:

- Typical limitations of these devices, in terms of: processing capability, persistent storage, runtime heap, battery lifetime, input methods and screen size, relative to stationary devices.
- High security risks and critical privacy requirements as any data stored in a mobile device, such as telephone numbers, birthdays and leisure time activities, are considered as private; even more than desktop computers, mobile devices are treated as personal [14].
- Great heterogeneity and non-interoperability in terms of hardware, Operating Systems (OS) and application software.
- Unreliable intermittent connectivity with low bandwidth.
- Highly demanding applications as applications intended to be executed in mobile devices should be designed carefully such that their problem space is decomposable and distributable among several devices [15].

More detailed descriptions of the mobile device limitations, and how they pose extra challenges when trying to apply the grid computing paradigm in the domain of mobile devices, are available in [8, 16].

However, [17, 18] necessitated the scaling of grids to both a large number of entities and to smaller devices. There are many indicators supporting this necessity. First, every measure of the capabilities of these devices including processing speed and memory capacity, is improving, and expected to continue, at exponential rate following Moore's law of increasing transistor density [19]. Second, the number of mobile devices in the world is escalating and expected to soon dominate the number of personal computers [20]. Indeed, the Wireless World Research Forum (WWRF) predicts that there will be 1000 wireless devices per person on average in 2017 [21]. Third, in many emergency situations, such as natural disasters and fire fighting, mobile devices might be the only accessible communication and computation tools. Fourth, the wireless connectivity and availability is improving as seen in current 3G networks. Fifth, it is difficult to materialise the NGG and AmI visions [18], where humans are surrounded by computing and networking technologies unobtrusively embedded in their surroundings, without utilising personal mobile devices. More details about cases against, and other supporting mobile devices in grid computing, are discussed in [22].

3.5.2 What Roles Can Mobile Devices Play?

Generally, two approaches have emerged in utilising mobile devices in grid environments. In the first approach, mobile devices serve as interfaces to stationary grids to send requests and receive results. Hence, a mobile device is merely playing the role of a resource consumer. Sometimes this approach is labelled as “mobile access to grid infrastructure” [15] or simply a “Mobile Access Grid”. In the second approach, mobile devices actively participate in the grid by providing computational or data services. Hence, a mobile device can play the roles of both a resource provider and resource consumer. This approach is what is usually referred to as a “Mobile Grid” [23]. A special case of Mobile Grids (and of Ad hoc Grids also) is identified in section 2.4.3.2.1 where all grid nodes are mobile in which the grid is labelled as a “Mobile Ad hoc Grid”.

3.5.3 Can Mobile Devices be Integrated in Grids?

If mobile devices are to be utilised, will they be integrated in current grid infrastructures or they will have their own?

Both approaches are available. In Mobile Grids and Mobile Access Grids, the most common approach is to integrate mobile devices with the grid infrastructure using a proxy between the stationary grid and mobile devices [24, 25]. In [14] caches are suggested to cope with the disconnectivity problem of mobile devices where operations on files are logged then automatically applied when a client reconnects. In Mobile Ad hoc Grids, grid nodes usually exchange services in a pure P2P scheme [26]. This can be done using intelligent agents [27, 28] or a mobile grid middleware system [29].

3.6 PM-Grid Design

The NGG vision has placed scalability, openness to wider user community, pervasiveness and ubiquity, transparency and user-centricity among its top desirable properties. Therefore, they are considered as the main non-functional requirements of PM-Grids. However, as stated in [30]:

> existing third generation GRID technology will not satisfy the requirement, and even great extensions to it will not satisfy the requirement. The way forward is to design an architecture based on the properties of NGG and implement it.

Hence, PM-Grid design has not adopted any of the already available grid architectures. Instead, the design is based on PN architecture and as a natural extension to them, seeing that scalability, pervasiveness and ubiquity, transparency and user-centricity have been explicitly addressed in their design [31]. A PM-Grid can be viewed as a superset of PNs. It is a PN with additional resources for sharing: CPU cycles and run-time memories, which allow for additional public and private services. This section starts with reviewing the architectural design of PNs, then builds up on this to arrive at an architectural design for PM-Grids.

3.6.1 PN Architecture

The PN concept and challenges have inspired many European Information Society Technology (IST) projects, such as My Personal Adaptive Personal Global Net (MAGNT) [5], MAGNET beyond [6] and Power Aware Communications for Wireless OptiMised personal Area Networks (PACWOMAN) [32], as well as the Dutch projects Personal Networks at Home (PN@home) [33] and the Personal Network Pilot 2008 (PNP2008) [34]. These projects have had an impact on the maturity of PNs design.

3.6.1.1 Layered View

Basically, as shown in Figure 3.6, a PN is composed of three abstraction levels: connectivity level, network level and service level. They are briefly outlined in following sections.

3.6.1.1.1 Connectivity Level

In the connectivity level, devices are grouped into various radio domains based on their radio interfaces. A radio domain is a group of devices with a common radio interface, a single Medium Access Control (MAC) mechanism and in direct communication range of each other.

3.6.1.1.2 Network Level

In the network level, devices within radio domains identified in the connectivity level are grouped into clusters based on a pre-established trust relationship. This trust relationship is very important to differentiate between personal nodes and devices and foreign nodes and devices. It is important to note that this trust relationship does not take into account devices owned by the user only but also other devices with long-term trust relationships such as family devices and devices from one's employment. The main function of this level is to separate communications among nodes of the same user from communications of other nodes.

3.6.1.1.3 Service Level

The service level is the highest level in the PN architecture. It contains all services offered by nodes in the Network Level. There are two types of services; public and private services. Public services are offered by both foreign and personal nodes and can be consumed by both. On the other hand, private services are offered and consumed by personal nodes only. While private services require establishing a long-term trust relationship, pubic services require a short-term trust relationship only. The service level contains all protocols related to service discovery and name servers [35].

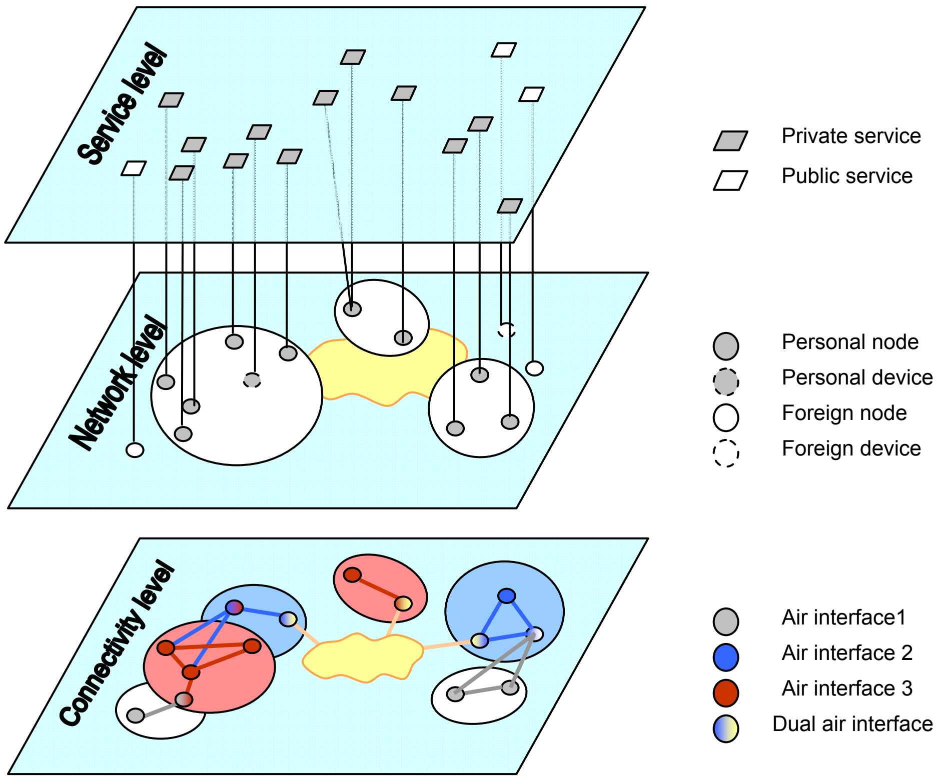


Figure 3.6: PN Layered View [35]

3.6.1.2 Detailed Architecture

Figure 3.7 shows the main elements of a PN. From an architectural point of view, a PN consists of the following elements:

- A PAN with the owner at its core: a set of personal nodes and devices around a person sharing a common trust relationship and communicating with others without relying on any foreign nodes or devices.
- Clusters: a cluster is a set of personal nodes and devices that share a common trust relationship and can communicate with each other without relying on any foreign nodes or devices.
- PN nodes: In each cluster/PAN, PN nodes communicate with each other using the IP protocol. PN nodes have multiple air interfaces to connect to other PN nodes and devices.
- PN devices: PN devices are devices that do not have IP capabilities. They are connected to other PN nodes and devices via a PN node.

- Gateway nodes: A personal node in a cluster/PAN does not operate in a stand-alone network; it needs to communicate with other nodes in remote clusters. Therefore, a gateway node with special features and functionalities, such as local storage and multiple network interfaces, address translation, tunnels set up and maintenance, traffic filtering among others, is employed to link PN nodes to remote and foreign nodes. Gateway nodes are usually selected as powerful devices as their tasks are quite load intensive.
- The PN agent: For gateway nodes to locate other gateway nodes in remote clusters and establish tunnels, the PN agent is used to provide additional services, such as naming and service discovery. The PN agent serves also as the entry point for PN to PN communication. It is important to add that the PN agent is not a device or node; rather it is a concept that might be implemented in different approaches.
- Interconnecting structure: A collection of overlapping networks of various technologies.

Obviously, a key element of a PN is the PN Provider (PNP) which offers the PN services. It provides the operational environment to manage users, services, content and network related issues [36].

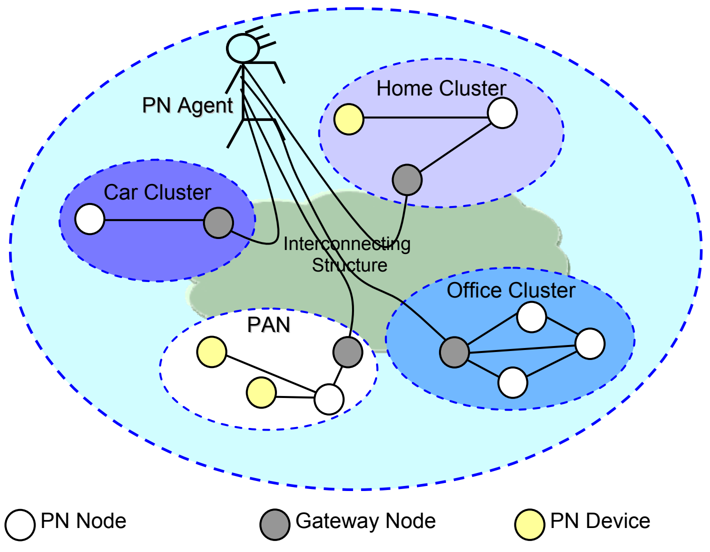


Figure 3.7: PN Detailed View

3.6.2 PM-Grid Architecture

It is important to note that the PN abstract levels and main elements presented in section 3.5.1 are part of the PM-Grid architecture. However, to avoid repetition in this section, only additional levels and elements that are required for the PM-Grid architecture are included.

3.6.2.1 Abstract Layered View

The PM-Grid architecture is based on the three levels PN architecture proposed by the MAGNET project [5]. An additional level is introduced between the network and service levels, namely the PM-Grid level. Hence, The PM-Grid architecture is composed of four abstract levels: the connectivity level, network level, PM-Grid level and the service level as shown in Figure 3.8. These levels act as a middleware system offering an abstraction over physical devices.

3.6.2.1.1 PM-Grid Level

The added PM-Grid level serves as a virtualisation layer to hide the complexity of harnessing the heterogeneous underlying computational resources from the end user. In this level, resources available from the network level are grouped into two main categories: personal resources residing inside the PM-Grid, and foreign resources residing outside the grid.

Personal resources are grouped into larger virtual resources based on the type of functionality they provide such as CPU cycles, storage, address book and printing. The aim is to allow personal users to submit service requests, for example a request for CPU cycles and memory to execute a computational job, from any device available within their trusted PNs without being concerned about where/when/how these requests are executed.

To achieve this goal, the grid level should provide an efficient resource scheduler. The scheduler is responsible for automatically decomposing, allocating and executing jobs, then finally composing final results, making them ready to the end user. The scheduler should be lightweight, self-managed and adaptive to cope with the dynamic nature of the PM-Grid environment. A detailed design of such a resource scheduler (HoPe) is presented in Chapter 5.

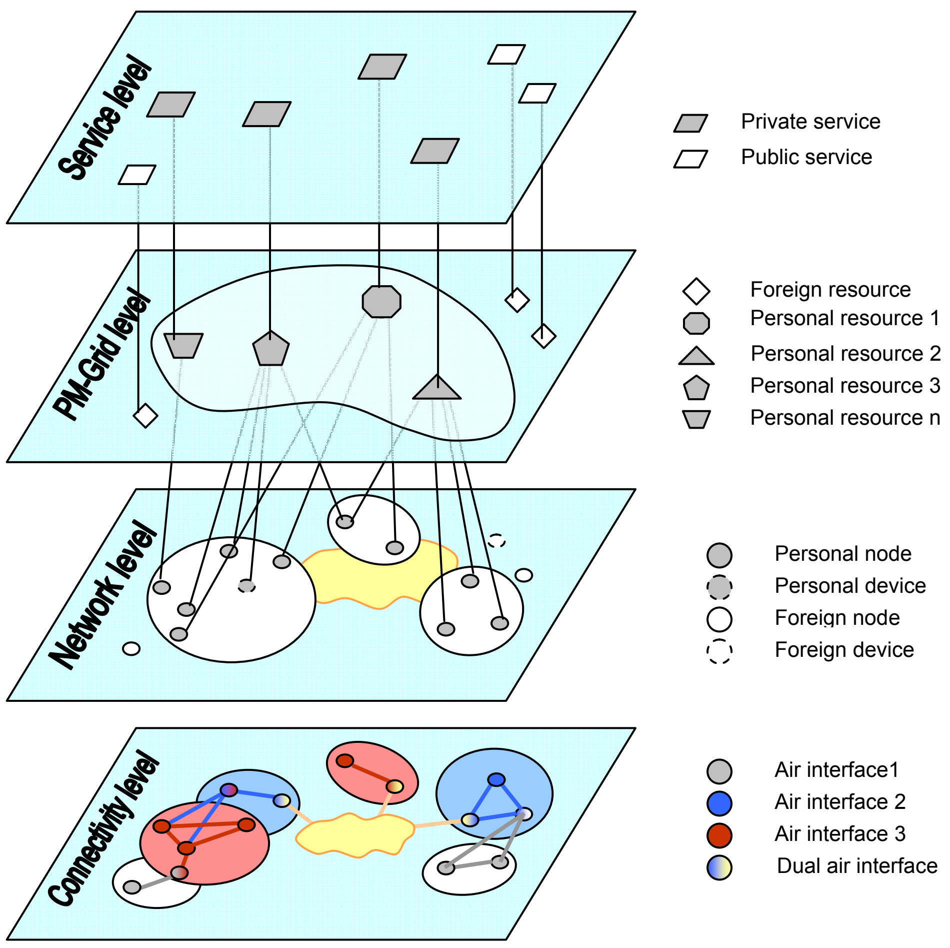


Figure 3.8: PM-Grid Layered View

3.6.2.2 Detailed Architecture

A PM-Grid consists of groups of devices which are usually owned and utilised by the same person. All these devices are connected via a well secured network PN. Issues related to connectivity are tackled in the PN connectivity level. Issues related to security and clustering are all handled at the PN network level, while issues related to presentation and quality of services are dealt with at the PN service level.

Thus, basically, the key missing functional component after superimposing grid functionality on top of a PN is a resource management system for the newly added grid resources represented by CPU cycles and runtime memories, as these resources require special handling to jointly execute computational jobs in PM-Grids. The main functions of this resource management system is to decompose parallel jobs, if possible, into smaller tasks that can be accommodated by mobile devices, then mapping these tasks to proper resources and, after execution, composing final results sending them back to clients.

Therefore, as shown in Figure 3.9, from an architectural point of view, a PM-Grid includes, apart from the PN architectural elements, three functional elements: clients, workers and spaces.

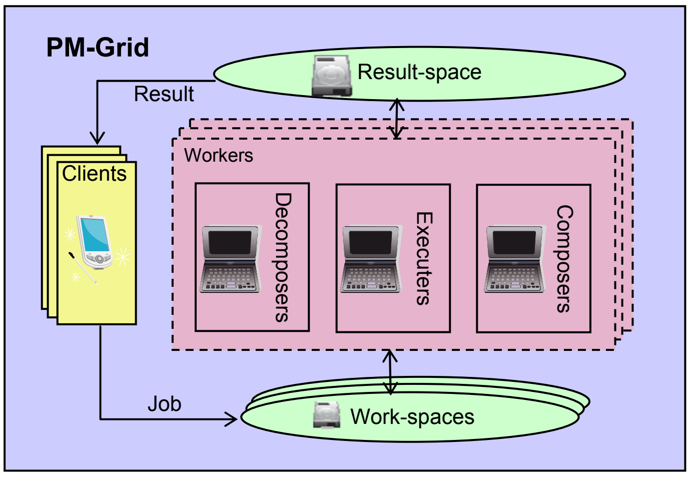


Figure 3.9: PM-Grid Detailed View

3.6.2.2.1 Clients

Clients represent the first category of PM-Grid elements, consisting of a set of mobile devices, such as mobile phones, usually within the PAN, that are highly dynamic and considerably limited in terms of processing power and network bandwidth. This set can send requests for executing simple jobs or complex computational jobs that are stored elsewhere, to more capable devices in PM-Grids.

3.6.2.2.2 Workers

Workers represent the second category of PM-Grid elements which consists of a set of devices, such as laptops, that can be mobile but are less dynamic and have better computing resources than clients. These devices can jointly complete computational jobs. They are divided further into:

- Decomposers: The exchange of large processing jobs in an environment of limited resources and network bandwidth increases the power consumption and slows the communication. Furthermore, PM-Grid devices may not be able to accommodate such large jobs. To tackle this problem in PM-Grids, decomposers are introduced. A decomposer is a specialised program that tests if a job might be executed in parallel. If so, it divides it into independent tasks of lower granularities.

- Executers: These are computing elements capable of executing the actual computation logic encapsulated in a job.
- Composers: Since jobs are decomposed into smaller tasks, and each task is executed independently of other tasks within the same job, there is a need to aggregate results produced after running these tasks. Composers are elements running a specialised program that compose all partial results related to a certain job into a final result to be sent to the requesting client.

3.6.2.2.3 Spaces

Spaces represent the third category of PM-Grid elements which consist of a set of static storage-rich devices mainly at home or the office, such as desktops. Clients and workers communicate with each other using these spaces which serve basically as simple shared memories for buffering. The use of a buffering technique is important in mobile environments to reduce the impact of frequent disconnections. The idea of spaces is based on Tuple-spaces first realised in the Linda system language [37]. A Tuple-space is a form of independent associative memory. For example, consider a group of processors that produce pieces of data and a group of processors that consume the data. Producers post their data to the space, consumers retrieve data from the space that matches certain criteria. In PM-Grids, there are two types of spaces:

- Work-spaces: Work-spaces are multiple pools of jobs sent from clients. Executers access these pools, hunting for tasks to execute.
- The result-space: the result-space is a large pool holding results that are generated by executers.

Basically, two approaches are available to organise spaces. A centralised approach with a single large space, this approach has a great impact in simplifying scheduling. However, managing such a space is usually rather a challenging problem due to its massive size. Additionally, this centralised space could become a bottleneck and represent a single point of failure. The other approach is to have multiple spaces in decentralised distributed fashion. While this approach aims to solve the main disadvantages of the centralised approach, it inherits the known disadvantages of decentralised schemes represented by performance degradation and poor coordination which usually lead to a load imbalance problem.

Therefore, in this thesis a new approach has been followed to avoid the shortcomings associated with previous approaches. The PM-Grid design is based on multiple independent work-spaces, where tasks to be executed are placed, as these spaces do not require coordination among them, as well as a single result-space where all results are buffered before being finally composed and sent to clients. The bottleneck problem in this case is easier to solve as the result-space is considerably smaller in size and lighter in traffic volume than a centralised space requiring much less management responsibility.

3.6.2.2.4 Device Roles in PM-Grids

The special organisation for distributing the system functionality among multiple agents (workers) with a single target pool (result-space) and multiple job sources (work-spaces) is inspired by the way honeybees are organised in a colony, as explained in Chapter 5.

In Figure 3.10 the hierarchal relationship between the main elements of a PM-Grid is illustrated. Although, each element had been defined earlier as a set of devices, an element actually represents a logical role which is a functionality that can be added to any device in a PM-Grid, based on its capabilities. Roles are “upward compatible” where workers can act as clients while spaces can act as workers and clients as well.

During initialisation, each device is assigned an initial role based on its score in the Device Score (DS) formula:

$$DS = w_1A_1+w_2A_2+w_3A_3+\ldots+w_nA_n \qquad (3.1)$$

where $A_1, A_2, A_3,\ldots,A_n$ are the set of static normalised attributes relevant to the device performance, such as CPU speed, memory size, network bandwidth, immobility and remaining power in the battery . The weighting coefficients $w_1, w_2, w_3,\ldots,w_n$ are used to describe the relative importance of the different device attributes in each role, subject to:

$$\sum_{i=1}^{n} w_i = 1, \qquad (3.2)$$
$$w_1, w_2,\ldots,w_n \geq 0$$

At the operation time, a device might be promoted (assigned a higher role in the PM-Grid roles hierarchy) based on the device score in the DS formula after substituting $A_1, A_2, A_3,\ldots,A_n$ by the device dynamic attributes such as current CPU load, available memory and battery consumption. For instance a laptop with a low battery might be promoted to a client.

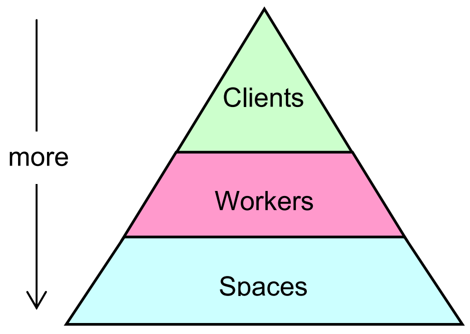


Figure 3.10: Role Hierarchy in PM-Grids

There are several other techniques to classify devices based on their capabilities, other than the weighting formula described above, for instance, semantic and ontology [38] as well as proxy-based solutions [39]. However, this issue of categorising devices based on their capabilities is not among the main concerns of this research. Therefore, for simplicity during the modelling of PM-Grids, roles are assigned manually to devices on their initialisation except executers and composers which exchange their roles automatically during running time based on their current and system contexts, as described in Chapter 5.

Device roles in PM-Grids are logically separated but physically may not be, which means that there might be more than one of these elements located within one physical device. In relation to PN elements, spaces are more likely to be located in gateway nodes as these nodes are usually powerful stationary devices such as desktops. The most powerful stationary device, in which usually the PN agent is located, is specifically chosen to accommodate the result-space. Workers are located in PN nodes as these devices usually have reasonable processing capabilities and multiple air interfaces such as laptops. Clients are located in PN devices which usually have the least resources. The placement of PM-Grid elements in relation to PN element is summarised in Table 3.1.

Table 3.1: Placement of PM-Grid Elements

PM-Grid elements	PN elements
Spaces	Gateway nodes
Workers	PN nodes
Clients	PN devices

3.7 Related Work

Connecting distrusted devices owned by an individual, or a group of individuals, and allowing them to share network resources is not the core of PM-Grids; PNs [5], PN Federation (PN-F) [6, 40], Personal Grid (PG) [41-44] and Personal distributed Environment (PDE) [45, 46] have been already proposed for this purpose. Allowing mobile access to grid systems is also not the core of PM-Grids; the Akogrimo project [23] has already addressed this issue. The novelty of PM-Grids is in superimposing computational grid functionalities on top of networked resource limited devices, whether they are mobile or stationary, and making the grid functionality available at personal users' hands. This section places PM-Grids amongst the above-mentioned projects and highlights the main similarities and differences.

3.7.1 PN and PN Federation

A PN offers a secure environment for a personal user to share network resources among his/her own devices. In MAGNET Beyond [6] and PNP2008 [34] the concept of PNs is extended into PN Federation (PN-F or Fednets), a secure cooperation between PNs of different users for a specific common purpose [40]. However, both PN and PN-F are concerned with sharing network resources such as data and peripherals rather than computing resources such as CPU cycles and runtime memories. Additionally, PN-Fs are formed only on demand for temporal situations; once the task is completed the network dissolves. On the other hand, PM-Grids are mainly concerned with sharing computing resources, and are set on a long-term basis for long-term goals.

3.7.2 Mobile Grids

The Akogrimo (Access to knowledge through the grid in mobile world) project [23] is the first IST project that explicitly targets Mobile Grids. While both Akogrimo and PM-Grid are concerned with integrating mobile devices in grid environments, Akogrimo is designed specifically for people in an enterprise domain, rather than for individual users in PM-Grids. The architecture of Akogrimo is based on an Enterprise Network which is built out of a consortium of enterprises in contrast to a PN underlying a PM-Grid which belongs to a single user. Additionally, mobile devices serve only as entry points to the grid in Akogrimo while they can participate actively in PM-Grids.

3.7.3 Personal Grids

A framework for a Personal Grid constructed over personal desktop computers is proposed in [41]. The framework consists of a two level hierarchal scheduling scheme where a super-node distributes jobs among clusters. Then, a master node in each cluster distributes the load among workers in FIFO style. The PM-Grid is different in that it extends the grid platform to mobile devices. Additionally, it has a distributed adaptive self-control scheduling scheme with no central entity, at the grid or cluster level, such as a super- or a central-node, making the scheduling decision.

The VEGA Grid project [42-44] has also proposed a framework for a Personal Grid (PG) to allow the integration of desktop computers into a “Global Grid System”. In this platform mobile devices are also used only as entry points to the grid. The PG aims primarily to establish a P2P platform for file sharing rather than processor sharing.

3.7.4 Personal Distributed Environment

In [45, 46] a Personal Distributed Environment (PDE) is proposed to allow a personal user to access his/her personal devices over heterogeneous networks to gain access to file sharing services such a global address book and the delivery of rich multimedia content. Again the main concern here is data communication rather than computations.

3.8 Conclusion

In this chapter PM-Grids were introduced as grid environments owned, constructed and utilised by personal users. They have the potential to scale the grid entities (service consumer and providers) to individuals and small size organisations. They also have the potential to widen the grid application areas to span more geographical and social settings than ever before.

An abstract layered architecture and a detailed inside view for PM-Grids based on PNs architectures were presented in this chapter. Furthermore, a lightweight, self-managed and adaptive scheduler was addressed as the core component of a middleware system for PM-Grids to cope with the dynamic nature of the environment.

Comparing PM-Grids to available grid projects shows that PM-Grids are the first to target both mobile devices and individual users at the same time and to offer file sharing as well as computational functionalities.

3.9 References

[1] J. Reimer, (2005, Dec. 14). Total share: 30 years of personal computer market share figures. ars technical [online]. Available: http://arstechnica.com/articles/culture/total-share.ars, [accessed Feb. 2, 2010].

[2] king computer [online]. Available: http://www. www.king-computer.com, [accessed Feb. 2, 2010].

[3] IEEE 802.15 Working Group for WPAN [online]. Available: http://ieee802.org/15/, [accessed Feb. 2, 2010].

[4] R. C.Braley, Ian C. Gifford, and Robert F. Heile, "Wireless personal area networks: an overview of the IEEE P802.15 working group," *SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 4, pp. 26-33, 2000.

[5] My Personal Adaptive Global NET (MAGNET) (IST 507102) [online]. Available: http://www.ist-magnet.org, [accessed Feb. 2, 2010].

[6] IST.MAGNET Beyond (IST-FP6-IP-027369) [online]. Available: http://www.magnet.aau.dk, [accessed Feb. 2, 2010].

[7] M. Waldburger and B. Stiller, "Toward the mobile grid: Service provisioning in a mobile dynamic virtual organization," in *Proc. 4th ACS/IEEE AICCSA-06,* 2006, pp. 579–583.

[8] Z. Li, L.Sun and E. C. Ifeachor, "Challenges of mobile ad-hoc grids and their applications in e-healthcare," *in Proc. 2nd CIMED2005*, Lisbon, Portugal.

[9] Global Health Smart Card [online]. Available: http://www.healthsmartcard.net/, [accessed Feb. 2, 2010].

[10] E-Health-Insider [online]. Available: http://www.e-health-insider.com/news/item.cfm?ID=2470, [accessed Feb. 2, 2010].

[11] Medical Office Online [online]. Available: http://www.medicalofficeonline.com/, [accessed Feb. 2, 2010].

[12] B. Crounse, (2006, Jul. 12). The future with electronic medical records: Effective, flexible, affordable, Microsoft [online]. Available: http://www.microsoft.com/industry/healthcare/providers/businessvalue/housecalls/clinicalworkflow.mspx, [accessed Feb. 2, 2010].

[13] San Diego Supercomputer Center [online]. Available: http://www.sdsc.edu/, [accessed Feb. 2, 2010].

[14] J. Roth and C. Unger, "Using handheld devices in synchronous collaborative scenarios," In *Personal Ubiquitous Comput.*, vol. 5, no. 4, pp. 243-252, 2001.

[15] D. C. Marinescu, G. M. Marinescu, J.Yongchang, L. Boloni and H. J. Siegel, "Ad hoc grids: Communication and computing in a power constrained environment," in *Proc. IEEE Int. Performance, Comput. Commun. Conf.,* 2003, pp. 113-122.

[16] Bhagyavati and S. Kurkovsky, "Emerging issues in wireless

computational grids for mobile devices," in *Proc. 8th World Multiconf. SCI-2004*. Available: http://www.cs.ccsu.edu/~stan/research/Grid/PubsSCI2004.pdf, [accessed Feb. 2, 2010].

[17] I. Foster and C. Kesselman, Eds., *The Grid2: Blueprint for a Future Computing Infrastructure*. San Francisco: Morgan Kaufmann, 2003.

[18] Expert Group, "Next generation grids2: Requirements and options for European grids research 2005-2010 and beyond," Expert Group Rep., Jul. 2004 [online]. Available: http://www.semanticgrid.org/docs/ngg2_eg_final.pdf, [accessed Feb. 2, 2010].

[19] K. Michael, (2003, Feb. 10) Moore's law to roll on for another decade, *cnet news* [online]. Available: http://news.cnet.com/2100-1001-984051.html, [accessed Feb. 2, 2010].

[20] Ipsos Insight, "Mobile phones could soon rival the PC As world's dominant Internet platform," Market study rep., summary, Apr. 2006 [online]. Available: http://www.ipsos-na.com/news/pressrelease.cfm?id=3049, [accessed Feb. 2, 2010].

[21] N. Jefferies, "Global vision for a wireless world," 18th meeting of WWRF, 2007, Helsinki, Finland.

[22] T. Phan, L. Huang and C. Dulan, "Challenge: Integrating mobile wireless devices into the computational grid," in *Proc. 8th Annu. Int. Conf. Comput. Networking,* 2002, pp. 271-278.

[23] Akogrimo [online]. Available: http://www.mobilegrids.org/, [accessed Feb. 2, 2010].

[24] S. Kurkovsky, Bhagyavati, and A. Ray, "A collaborative problem-solving framework for mobile devices," in *Proc. 42nd ACM-SE*, 2004, pp. 5-10.

[25] A. Hampshire, "Extending the open grid services architecture to intermittently available wireless networks," in *Proc. UK eScience All Hands*, 2004.

[26] L. Lima, A. Gomes, A. Ziviani, M. Endler, L. Soares and B. Schulze, "Peer-to-peer resource discovery in mobile Grids," in *Proc. 3rd int. workshop MGC '05*, 2005, pp. 1-6.

[27] S. Kurkovsky and Bhagyavati, "Modeling a computational grid of mobile devices as a multi-agent system," in Proc. IC-AI'03, 2003, pp. 36-44.

[28] Kurkovsky, Bhagyavati and A. Ray, "Modelling a grid-based problem-solving environment for mobile devices," *J. Digital Inform. Manage.,* vol. 2, pp. 109-114, 2004.

[29] A. Sajjad, H. Jameel, U. Kalim, S. Han, Y. Lee and S. Lee, "AutoMAGI - an autonomic middleware for enabling mobile access to grid infrastructure," in *Proc. 2005 ICAS-ICNS,* pp. 73-79.

[30] K.G. Jeffery, "Next generation grids for environmental science,"

Environmental Modelling & Softw., vol. 22, no. 3, pp. 281–287, 2007.

[31] B. Jiang, V. Kaldanis, A. Markopoulos, M. Monti, R. Prasad, D. Saugstrup, B. Jiang, V, Kaldanis, A. Markopoulos, M. Monti, R. Prasad, D. Saugstrup, N. Schultz and K.E. Skouby, "User requirements & demand for services and applications in PNs," presented at IST mobile & wireless communication summit, Lyon, France, 2004.

[32] PACWOMAN. (IST-2001-34157) [online]. Available: http://www.imec.be/pacwoman/, [accessed Feb. 2, 2010].

[33] The Dutch Freeband Communications Project QoS for PNs at home [online]. Available: http://www.qos4pn.irctr.tudelft.nl/index.htm, [accessed Feb. 2, 2010].

[34] The Dutch Freeband Communications Project PNP2008 [online]. Available: http://www.freeband.nl, [accessed Feb. 2, 2010].

[35] M. Jacobsson, J. Hoebeke, S. de Groot, A. Lo, I. Moerman and I. Niemegeers, "A Network Layer Architecture for Personal. Networks," presented at MAGNET Workshop, Shanghai, China, Nov. 2004.

[36] F. Hartog and M. Peeters, "A concrete example of a personal network architecture," in *Proc. 3rd IEEE CCNC2006*, pp. 514-518.

[37] D. Gelertner, "Generative communication in Linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, pp. 80-112, 1985.

[38] A. Bandara, T. Payne, D. Roure, and G. Clemo, "An ontological framework for semantic description of devices," *In Proc. ISWC 2004*, Hiroshima, Japan.

[39] X. Sanchez-Loro, J. Casademont, J. L. Ferrer and J. Paradells, "A proxy-based solution for device capabilities detection," in *Proc. IASTED* , 2007 pp. 28-34.

[40] M. Ibrohimovna, and S. H. Groot, "Proxy-based Fednets for sharing personal services in distributed environments," *in Proc. 4th ICWMC*, 2008, pp.150-157.

[41] J. Han and D. Park, "A lightweight personal grid using a supernode network," in *Proc. 3rd Int. Conf. P2P2003*, pp. 168-175.

[42] W. Li, Z. Xu, B. Li, Y. Gong, "The Vega Personal Grid: A lightweight grid architecture," *in Proc. IASTED*, 2002, pp. 6-11.

[43] B. Li, W. Li and Z. Xu, "Personal Grid running at the edge of internet," *in Proc. 2nd GCC*, 2003, pp. 762-769.

[44] Z. Xu, L. Xiao and X. Liu, "Personal Grid," *in Proc. NPC*, 2007, pp. 536-540.

[45] D. Pearce, J. Dunlop and R.C. Atkinson, "Leader election in a personal distributed environment," in *Proc. IEEE 16th Int. PIMRC*, 2005, vol. 2, pp. 1307-1311.

[46] J. Dunlop, "The concept of a personal distributed environment," *Wireless Personal Commun. Int. J.*, vol. 42, no. 3, pp. 431-444, 2007.

Chapter 4
A Framework for Resource Scheduling

4.1 Introduction

In contrast to the scarcity of resources proposing surveys or taxonomies for emerging grids, as seen in Chapter 2, plethora of literature has proposed taxonomies for resource scheduling systems in distributed systems in general such as [1-7], and grid schedulers in particular such as [8-13]. These taxonomies collectively span nearly every single aspect related to resource scheduling. This abundance of resource scheduling taxonomies can be related to two main influences, namely, the maturity of research in the area of resource scheduling, and the critical role that resource schedulers play in many application areas including grid computing. On the other hand, two major problems have emerged as a consequence: first, there are scattered nomenclatures across the literature, and second, there are inconsistent and unclear definitions for many of the terminologies.

Therefore, the goal of this chapter is to present a framework for resource scheduling and to provide a unified presentation of the main nomenclatures from several previously published taxonomies indicating, when necessary, the different terminologies in use. In other words, this is an attempt to amalgamate the area of resource scheduling systems together under a common, uniform set of terminologies. The intention has been to provide a suitable framework for comparing, analysing and studying work in the area. The material in this chapter provides a level of detail and a unifying perspective that can help in future research in the resource scheduling field.

Section 4.2 defines the resource scheduling problem and sheds some light on its historical context. Section 4.3 lists some basic terminologies in the problem domain. In section 4.4, a framework is proposed for resource scheduling systems, and a unified taxonomy for the framework elements is proposed. In Section 4.5, the scheduling problem is viewed from the grid computing perspective indicating its special features and applying the proposed framework to three well established grid resource schedulers. Section 4.6 briefly concludes the chapter.

4.2 The Resource Scheduling Problem

In general terms, scheduling is a mechanism to allocate resources to jobs with the objective to optimise one or more performance measures, as illustrated in Figure 4.1. The mechanism belongs to a broader class of combinatorial search problems which are concerned with finding combinations of a discrete set of items that satisfy some specified constraints. The number of possible combinations grows exponentially with the size of the problem leading to potentially lengthy solution times and severely limiting the feasible size of such problems. Therefore it is among the most difficult of common computational problems, which are considered as NP-hard [14]

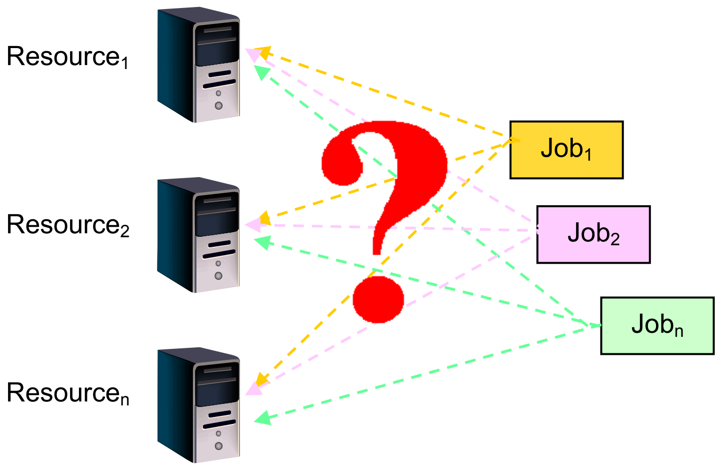


Figure 4.1: The Resource Scheduling Problem

The scheduling problem was initially identified during the 1950s, in operations research, industrial engineering and management. After that, in the 1960s, it was introduced to computer science in operating systems development. The problem started with simple forms that could be optimally solved using efficient algorithms. As time went by, the problem became more sophisticated hindering the search for efficient algorithms for many of its forms. By the advent of complexity theory [15] complex forms of the scheduling problem had been considered to be NP-hard, in the 1970s. Therefore, other directions to the problem were introduced during the 1980s: approximation and heuristic approaches as well as stochastic scheduling [16], as described in section 4.4.4.3.2 and section 4.4.4.2.1 respectively. In the early millennium, the swarm intelligence approach, presented in section 5.4.2, has emerged to suggest solutions to NP-hard scheduling problems based on techniques from social insects. Now, after sixty years, there is a solid body of knowledge in this field.

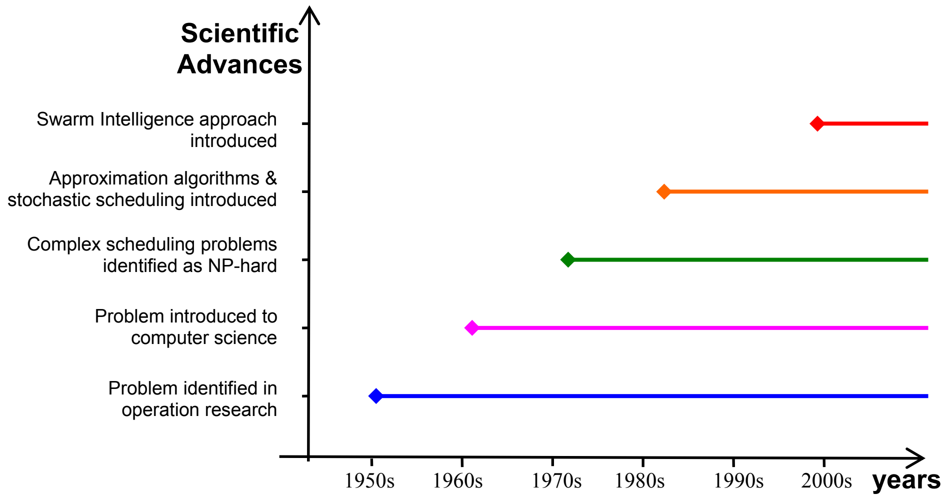


Figure 4.2: Scientific Advances in the Resource Scheduling Field

4.3 Basic Terminologies

There are some common terms used in the resource scheduling field. In this section they are briefly defined:

- A resource is anything that is required to carry on an operation, and includes such items as machines, processors and runways.
- A job is anything that consumes resources. It usually consists of a single set of multiple tasks. A job can be a manufacturing process, a computer program, a landing or take-off, etc.
- A task is an atomic operation to be performed on a resource.
- A performance metric, also known as the objective function, is the objective under consideration such as the minimisation of the makespan or maximisation of the throughput.

- A schedule is a mapping between tasks and resources.

In this chapter the two terms: resources and jobs are mainly used to refer to resources and jobs associated with computers such as processors and application programs. However, this does not mean that the materials presented in this chapter are restricted to this specification alone.

4.4 A Framework for Resource Scheduling

A shared characteristic among previous scheduling taxonomies was the vast number of nomenclatures they proposed; for instance in [9] eleven main nomenclatures were presented. Although this might help in detailed classification it complicates the taxonomy, entangles its nomenclatures and makes the search for common features more difficult. Therefore a common framework that identifies focal entities of resource schedulers and a taxonomy based on these entities would tackle such problems.

Basically, resource scheduling systems deal with four main entities: jobs, resources, performance metrics and a scheduler. In solving a scheduling problem, four questions are usually considered:

- How do resource characteristics affect the scheduling decision?
- How do job characteristics affect the scheduling decision?
- What performance measures should a scheduler use to determine the quality of a schedule?
- Which scheduler (policy, architecture and procedure) gives best (or good) results based on the previous three concerns?

This chapter presents a framework for resource scheduling, regardless of the problem domain, based on the above four elements: job model, resource model, performance metrics and scheduler model, as shown in Figure 4.3. It also presents a unified taxonomy, as shown in Figure 4.4, to describe the main features of these elements in an attempt to provide a unifying perspective that can help in designing and analysing resource schedulers. However, since the taxonomy is not intended to be comprehensive, it only drills down in special categories that are considered as important requirements for HoPe to present a clear background about its design features.

It might be reasonable to point to the difference between the taxonomy developed in this chapter and the comprehensive taxonomy presented in Chapter 2. The taxonomy presented in Chapter 2 is intended to serve mainly as a classifying tool. On the other hand, the taxonomy presented in this chapter with the proposed framework, is intended to assess mainly initial design stages to identify scheduler requirements and features. It can also assist in analytical studies for comparative purposes.

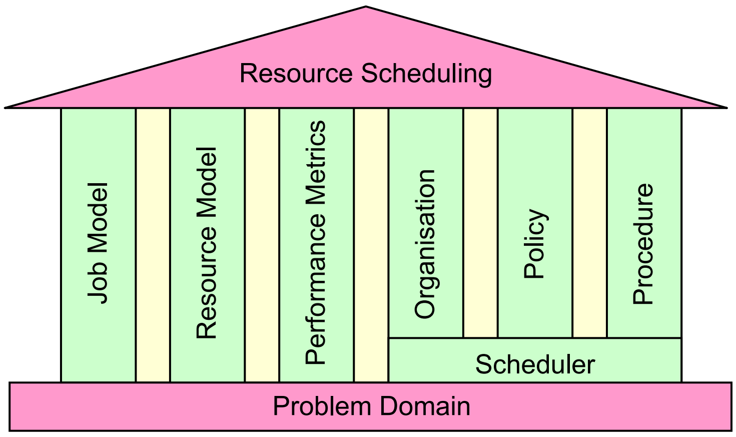


Figure 4.3: Resource Scheduling Framework

4.4.1 Resource Model

The characteristics of underlying resources are critical for making the scheduling decision. For a scheduler to make a decision it needs to know:

- Whether resources are of the same type, or of different types.
- The characteristics of each resource.

Accordingly, two main resource models are identified in the literature: parallel and dedicated resources, as illustrated in Figure 4.5.

4.4.1.1 Parallel versus Dedicated Resources

Parallel resources are capable of performing the same functions. They are categorised further based on their speed as identical, uniform and unrelated resources, as explained in section 4.4.1.1.1.

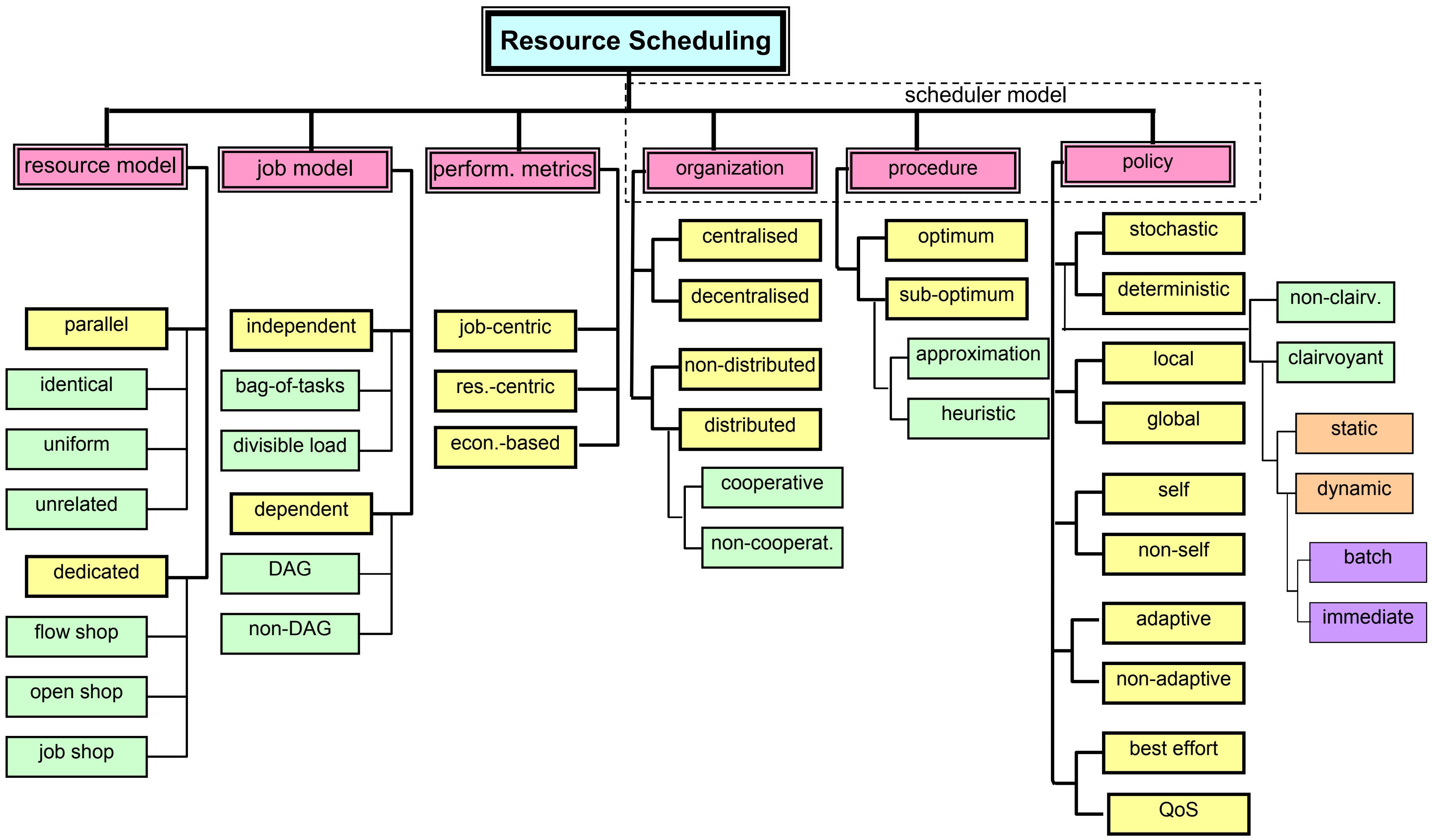


Figure 4.4: Unified Taxonomy for Resource Scheduling

In contrast, dedicated resources are specialised in executing certain tasks only. Three distinguished scheduling models are identified based on the order in which these tasks follow inside the system: flow shop, open shop and job shop scheduling models, as explained in section 4.4.1.1.2.

4.4.1.1.1 Identical, Uniform and Unrelated Parallel Resources

Identical resources are parallel resources with equal processing speeds. Uniform resources are parallel resources but with different processing speeds. However, the speed of each uniform resource is constant for all types of jobs. In contrast, each unrelated resource has a variant speed associated with each type of job.

4.4.1.1.2 Flow, Open and Job Shops Dedicated Resources

These three scheduling models are based on the order in which jobs visit dedicated resources. In the flow shop scheduling model, each job is executed on all machines following a certain order. In the open shop model, each job is processed once on each machine with no constraint about the order of processing. In the job shop model a job can be processed on more than one machine and has its own order in visiting machines.

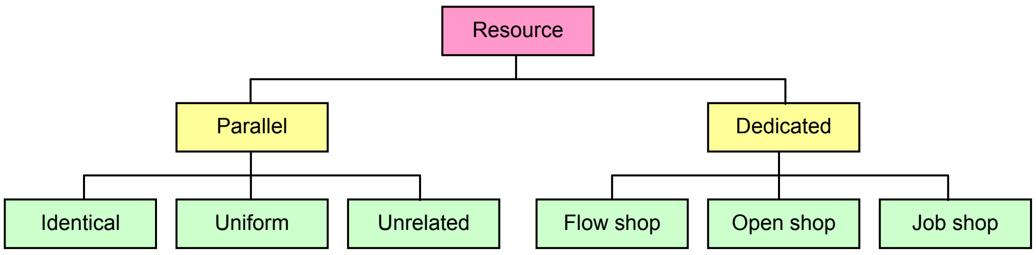


Figure 4.5: Resource Model

4.4.2 Job Model

The job model has a significant impact on the scheduling decision. For a scheduler to make a decision it needs to know:

- The characteristics of each job in terms of its internal structure.
- The amount and type of interaction it requires with other jobs or with the running environment.

Based on this information, jobs are classified into two main categories: non-independent jobs and independent jobs, as illustrated in Figure 4.6.

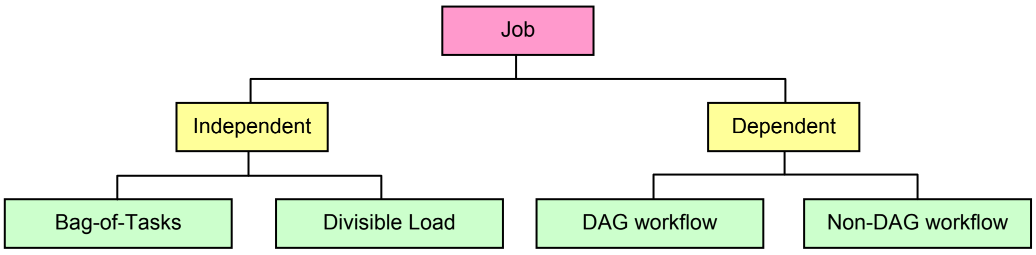


Figure 4.6: Job Model

4.4.2.1 Dependent Jobs versus Independent Jobs

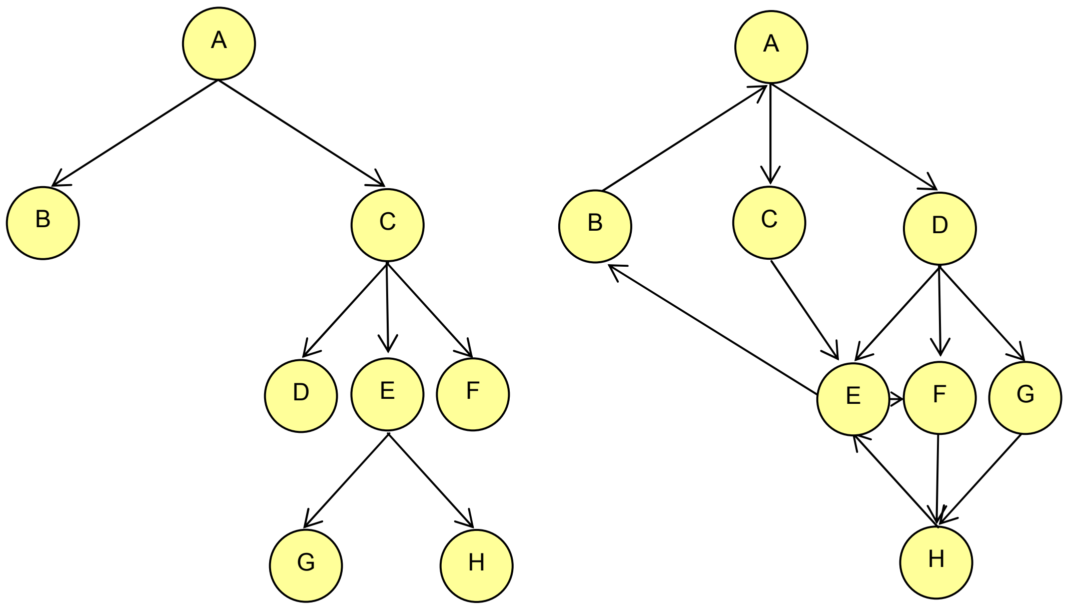
Dependent jobs, usually known as workflows, are coarse-grained applications constructed from a sequence of components (tasks). Tasks themselves are considered heterogeneous in nature; they might be sequential or parallel having different behaviour and resource requirements [17]. Workflows vary in their internal structure, and there are two categories: directed acyclic graph (DAG) workflows and non-DAG workflows, as described in section 4.4.2.1.1.

An independent job represents an application which is composed of a set of tasks with no communication, dependencies or synchronisation among them. These tasks can be executed in any order since each task does not require any input from any other task. In other words, the output of any task would never be fed to another task as an input. However, multiple tasks can share the same input file(s) and they may also share the same output file(s). These applications are easy to parallelise by decomposing them into multiple tasks of lower granularity. From a theoretical perspective, an independent job model is a generalisation of the pre-emptive execution model that allows for simultaneous execution of different parts of the same job on different machines [18].

Applications conforming to this model arise in many fields of science and engineering such as image processing, Monte Carlo simulations, data mining and database searching [19]. There are two possible models for independent jobs based on the task granularity: Bag-of-Tasks (BoT) and Divisible Load (DL), as described in section 4.4.2.1.2.

4.4.2.1.1 DAG Workflows versus Non-DAG Workflows

In DAG workflows the internal structure of a workflow is represented by a DAG. Nodes of the graph represent tasks while edges represent dependencies between tasks. The simplest workflow applications can be represented with a simple DAG in which tasks are performed in a specific linear order. At the second level of complexity are workflows that are modelled using non-linear DAG. Some scientific applications require an iteration structure; in this case, workflows are modelled with cyclic graphs and are called non-DAG workflows. In the most complicated level of workflows it is even difficult to find an appropriate graph model for the workflow. In this case, an application is modelled as a workflow of workflows [20].



(a) DAG Workflow **(b)** Non-DAG Workflow

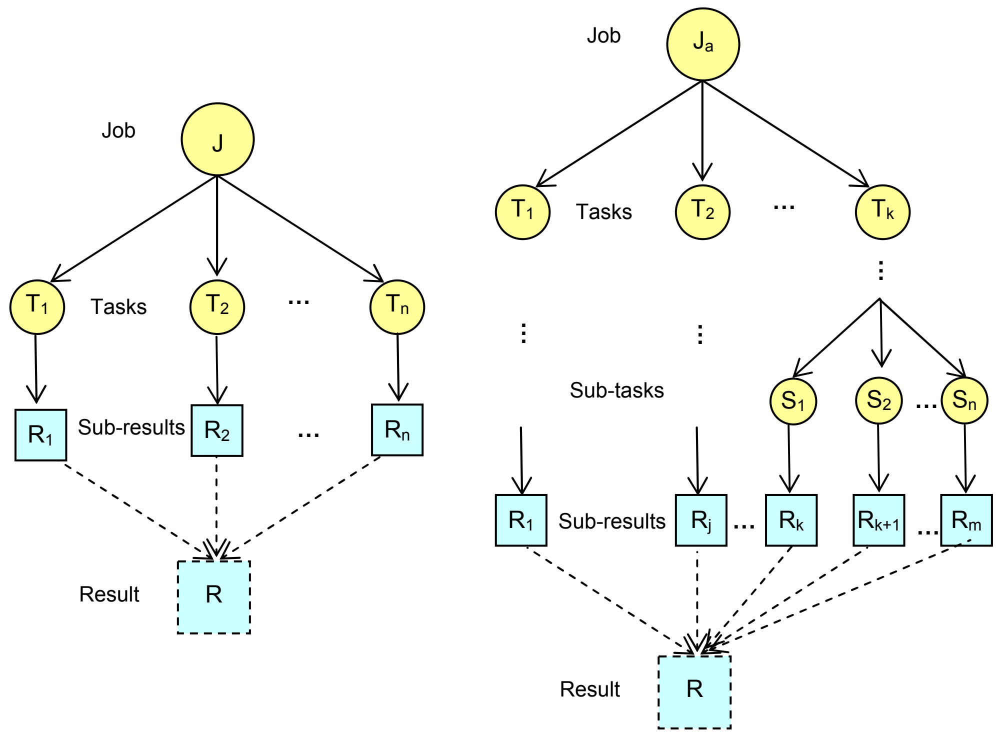
Figure 4.7: Dependent Job Example Models

4.4.2.1.2 Bag-of-Tasks versus Divisible Load

Independent jobs can be composed of coarse-grained components which are known as Bag-of-Tasks (BoT), or fine-grained components which are known as Divisible Loads (DL). However, some work in this area [21, 22] use the term divisible load to refer to both types with the former considered as modularly divisible and the latter as arbitrarily divisible.

BoT jobs are also known as parameter-sweep applications [23]. A BoT is a coarse-grained application consisting of computations that can be divided into a finite number of independent pieces (tasks). The number of tasks and the task size of each application are set in advance. In this case the scheduling problem is normally considered as a bin packing problem. This problem is considered to be NP-hard and is usually approached by means of heuristics [19].

DL applications, also known as fine-grained applications, consist of computations or loads that can be arbitrarily divided into independent chunks (tasks) [24]. This corresponds to a perfectly parallel job: any task can itself be further decomposed into independent sub-tasks. A DL model is an approximation of job models that are built out of a large number of identical, low granularity components [25]. It has the potential to provide a practical platform for scheduling in heterogeneous environments [26].



(a) BoT Job **(b)** DL Application

Figure 4.8: Independent Job General Models

4.4.3 Performance Metrics

Performance metrics, also known as scheduling objectives, can be viewed from two different and competing perspectives: the user or consumer perspective (Job-centric metrics) and the provider perspective (resource-centric metrics), as described in section 4.4.3.1.

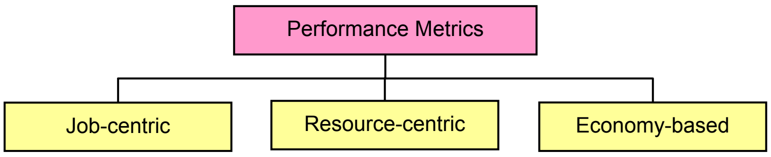


Figure 4.9: Performance Metrics

4.4.3.1 Job-Centric versus Resource-Centric Metrics

Job-centric metrics, also known as user-centric metrics, represent the user or consumer perspective. They seek to optimise the performance of each individual job, such as the turnaround time (also known as flow time, response time or completion time), which represents the time taken from when a job enters the system until it finishes execution. Job centric metrics are related to the system performance which encompasses how well system resources are being used for the benefit of each user of the system.

Resource-centric metrics, also known as provider-centric metrics, seek to optimise the system efficiency such as throughput, resource utilisation and makespan (the total time required for completing all jobs in a set). The system efficiency is concerned with how efficiently resources are utilised for the benefit of all users of the system, as well as the added overhead associated with the resource scheduling process [3].

As job-centric and resource-centric metrics are competitive, there are always tradeoffs to consider, therefore hybrid approaches such as economy-based metrics were proposed. Economy-based metrics consider both job (resource consumer) and resource (resource provider) perspectives at the same time but from the market economy point of view. For the market to be competitive, resource providers need to set reasonable prices to keep the supply of a service equal to its demand. However, applying these metrics requires that the whole system is built initially, with the economic model as a reference model [27].

4.4.4 Scheduler Model

A scheduler model describes the organisation, policy and procedure of a resource scheduler.

4.4.4.1 Organisation

Scheduler organisation means the way that entities involved in the scheduling process interact with each other. This organisation has a critical influence on the efficiency of the scheduling process. There are three main features which are used in the literature to describe the organisation of resource schedulers:

- Centralised versus decentralised.
- Distributed versus non-distributed.
- Cooperative versus non-cooperative.

However, some of these features are used interchangeably, ignoring the actual difference between them, as described in the following sections.

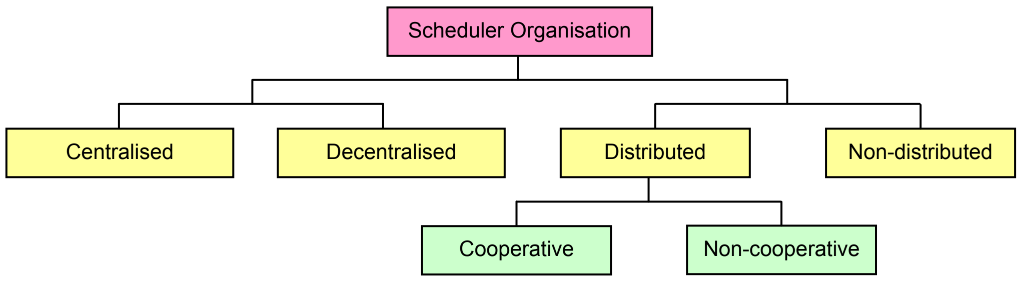


Figure 4.10: Scheduler Organisation

4.4.4.1.1 Centralised versus Decentralised

In centralised schedulers, a single entity has the authority to make the scheduling decision; it makes the decision for the whole system regarding who should run what and when. This organisation has the advantages of simplified management and deployment. Among the main disadvantages are the lack of fault tolerance, poor scalability and the difficulty in accommodating multiple policies.

In decentralised schedulers, the scheduling authority is shared among the multiple entities of a resource management system. This organisation eases scaling to large systems and is more fault tolerant if proper coordination is shouldered by the different schedulers.

There also exist hierarchical schedulers which are organised in multiple levels, so the higher level scheduler, also known as a meta-scheduler, controls larger sets of resources than lower level schedulers. Although this organisation addresses the scalability and fault tolerance issues, the problem of the multiplicity of scheduling policies is still unsolved. It also suffers from the added difficulty of coordinating schedulers in different levels [9].

4.4.4.1.2 Distributed versus Non-Distributed

In non-distributed schedulers, the responsibility for executing the scheduling policy physically resides in a single entity, whereas in distributed schedulers this responsibility is shouldered by physically distributed entities. It is important to note that the two terms, decentralised and distributed, are used interchangeably in the literature, while they actually refer to different aspects of the scheduling process: responsibility and authority. When the responsibility for making and carrying out policy decisions is shared among entities in a system, the scheduler is distributed. On the other hand, when the authority of making the scheduling decisions is distributed to the system entities, the scheduler is decentralised [3].

4.4.4.1.3 Cooperative versus Non-Cooperative

Distributed schedulers can be classified further, based on the way an individual processor makes decisions, while executing the scheduling policy, into: co-operative and non-cooperative schedulers. In non-cooperative schedulers, individual entities act alone as autonomous agents and arrive at the scheduling decision independently of the action of other entities in the system. In cooperative schedulers, each entity has the responsibility to carry out its own portion of the scheduling task, but all entities are working toward a system wide goal [3].

4.4.4.2 Scheduling Policy

A scheduling policy consists of a set of general features describing the scheduling process. However, these features are scattered in the literature with no clear definition for many of them. Furthermore, some features are used interchangeably while they actually describe different scheduling attributes. Therefore, this section presents a more comprehensive list of policy features with a clear definition of each:

- Stochastic versus deterministic.
- Clairvoyant versus non-clairvoyant.

- Static versus dynamic.
- Immediate versus batch.
- Adaptive versus non-adaptive.
- Local versus global.
- Self-scheduling versus external scheduling.
- Best effort versus QoS.

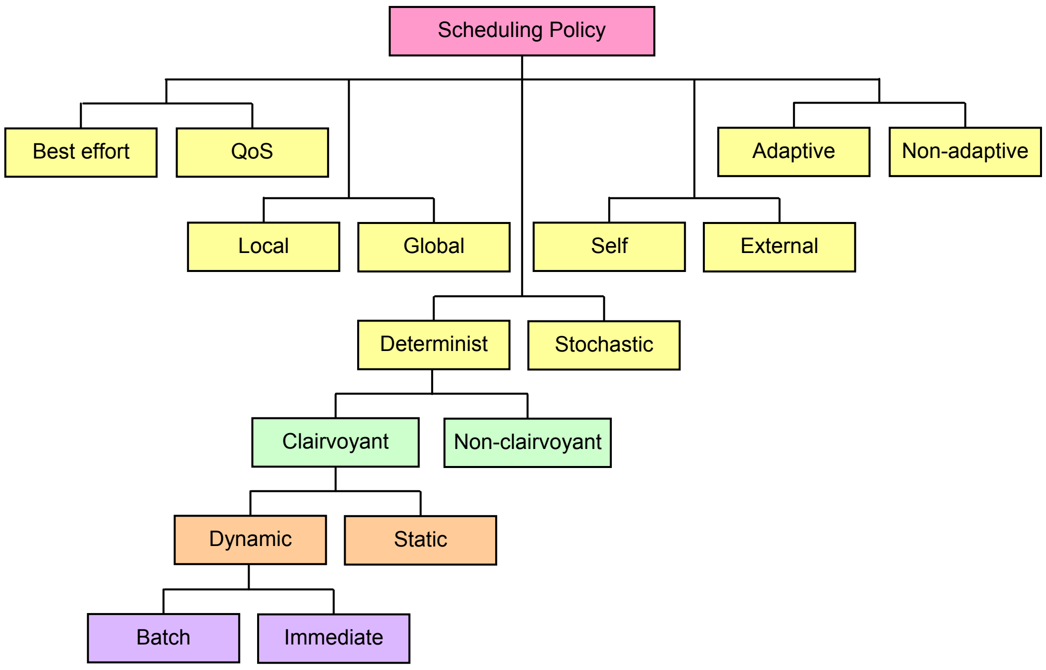


Figure 4.11: Scheduling Policy

4.4.4.2.1 Stochastic versus Deterministic

Based on the way information about jobs and resources is generated, one can differentiate between deterministic and stochastic policies.

Basically, stochastic means random. In other words, it is determined by chance. In stochastic scheduling, job information, such as the processing time, is unknown in advance, but it is known to be a random selection of a given probability distribution. The actual information only becomes known when the processing has been completed [16]. Stochastic scheduling is used where either the number of individuals is small or where there is reason to expect random events to have an important influence on the behaviour of the system [28]. Stochastic and non-clairvoyant scheduling, described in section 4.4.4.2.2 were introduced to deal with the uncertainty problem in job processing times.

In contrast, deterministic means that no job information is probabilistically determined. Deterministic scheduling takes no account of random variation and therefore gives a fixed and precisely reproducible result, quite the opposite to stochastic scheduling where different outcomes can result from the same initial conditions [28]. However, it does not require that all job information is known in advance. Rather, it also considers problems where some job parameters are unknown in advance [29], such as non-clairvoyant and dynamic scheduling.

4.4.4.2.2 Clairvoyant versus Non-Clairvoyant

Among the significant factors that affect the scheduling decision are the volume and type of information available to the scheduler. Greater volumes of information about jobs, such as the number of jobs, their processing times and release dates can result in an optimum schedule.

However, such information may not be available or may be too expensive to collect. Also, increasing the amount of information processed by a scheduler usually increases the time to produce a schedule [30]. Therefore, two contrasting scheduling policies can be addressed based on the availability, or necessity, of such information: clairvoyant and non-clairvoyant scheduling.

In a clairvoyant scheduling policy, it is assumed that job characteristics, such as execution time and release dates, are available to the scheduler before the scheduling decision takes place; that is, either before jobs enter the system (static scheduling) or just before starting their execution (dynamic scheduling), as described in section 4.4.4.2.3. This clairvoyant scheduling is usually what the classical scheduling theory considers and, with which almost all research in scheduling theory has been concerned [31]. However, this assumption is the strictest one in the scheduling theory and it has a great impact in limiting its practical application. “Indeed, this assumption is not valid for the most real world processors” [32]. In contrast, a non-clairvoyant scheduling policy assumes and requires no prior knowledge about job or resource characteristics. This information might only be available after a job has been executed.

It is important to note here the difference between the non-clairvoyant policies and dynamic scheduling policies, presented in section 4.4.4.2.3 which are usually confused in the literature. When job information is available to the scheduler before it starts running, it is said that the scheduling policy is dynamic. When job information is only available after the job is executed, this is called non-clairvoyant scheduling [31]. The difference in time, at when job information becomes available to the scheduler, between static, dynamic, and non-clairvoyant scheduling is illustrated in Figure 4.12.

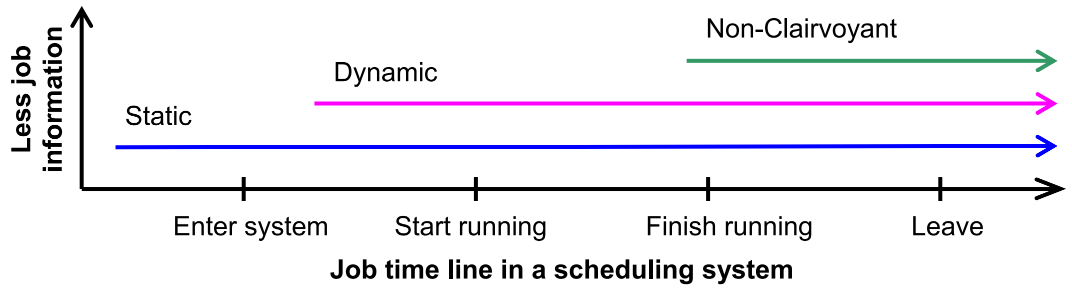


Figure 4.12: Static, Dynamic and Non-Clairvoyant Scheduling

4.4.4.2.3 Static versus Dynamic

Based on the time when job and resource information is available, clairvoyant scheduling policies can be considered as either static or dynamic.

In a static (also known as plan-ahead and offline) scheduling policy, information about jobs and resources are assumed to be available before jobs enter the system. However, this policy is not applicable when job or resource characteristics are not known in advance. In dynamic (also known as on-the-fly or online) scheduling policies, less information is known a priori. Job information is only available after entering the system and sometimes just before it starts execution [33]. Dynamic policies are classified further based on when the scheduling decision occurs into immediate mode and batch mode policies, as described in section 4.4.4.2.4.

4.4.4.2.4 Immediate versus Batch

Within the realm of dynamic scheduling policies, two approaches can be identified based on when the scheduling decision takes place: immediate and batch policies. An immediate mode policy maps a job to a machine upon task arrival, whereas a batch mode scheduling policy is event driven. So, when a specified condition is satisfied, such as a certain number of tasks, or a time period elapsed, scheduling occurs.

4.4.4.2.5 Adaptive versus Non-Adaptive

In an adaptive scheduling policy the scheduling algorithm or parameters are dynamically modified according to the change in the system state. In a non-adaptive scheduling policy, the current system state has no influence on the scheduling policy. The two properties, dynamic and adaptive, are often used interchangeably in the literature while they actually represent slightly different features [3]. In a dynamic policy, part of the information about jobs and resources is revealed dynamically thus schedules are generated in the same manner. However, this does not necessarily imply that the scheduling algorithm or parameters are dynamic as well, which is the case in adaptive scheduling policies.

4.4.4.2.6 Local versus Global

In general, decisions about mapping tasks to resources can be made at two levels: local level and global level. In a local scheduling policy, decisions are made based only on the job (sometimes a group of jobs or a sub-workflow) at hand. In a global scheduling policy, decisions are made based on all non-scheduled jobs (sometimes jobs not yet started or the whole workflow). The main advantage of global policy schedulers, also known as meta-schedulers, is that they can provide a better overall result. On the other hand, making the scheduling decision takes a much longer time than local policies. Thus the overhead produced by a global policy can reduce the overall benefit and possibly exceed its benefits [11].

However, there is no agreement about what is local and global scheduling. In [34] local scheduling is defined as the policy that considers one administrative domain only, such as a cluster, whereas a global scheduling policy considers multiple administrative domains. In [3] local scheduling is defined as the policy concerned with mapping jobs within one machine whereas global scheduling considers mapping in multiple machines. In this chapter, we follow the same approach as [11] in defining global and local scheduling policies.

4.4.4.2.7 Self-Scheduling versus Non-Self Scheduling

A non-self-scheduling scheme is what classical scheduling usually assumes where a dedicated system or authority is responsible for making the scheduling decision, implementing the scheduling policy and executing the scheduling procedure. This approach requires an external entity, to gather information about each node. This can have high security risks, involves a lot of message exchange and hinders each node from having its own policy

On the other hand, in self-scheduling policies processors do both duties of assigning jobs to themselves and executing them. Whenever a processor becomes free it picks from a shared job pool, a ready task whose predecessors (if any) are all completed according to a scheduling order [35]. There has been increasing interest in the self-scheduling scheme using different approaches such as intelligent agents, market model and swarm intelligence [36]. More about self-scheduling schemes is presented in [37-39].

4.4.4.2.8 Best Effort versus QoS

A schedule might offer the best performance for a job at its start but over time other jobs may introduce load into the system, or job requirements may change. To sustain good performance, high Quality of Service (QoS) and fault tolerance for long running jobs and real-time applications, schedulers usually include additional features, such as pre-emption, rescheduling, co-scheduling and resource reservation, to support such applications. These features are outlined as follows:

- Pre-emption: a pre-emptive scheduling policy may block a job after it started execution and resume it later in the same or a different machine.

- Rescheduling: a rescheduling policy allows changing the machine in which a job is running (migration). It also allows swapping between jobs when a certain event occurs such as new job arrival or machine down.

- Co-scheduling**:** In a co-scheduling policy, related jobs of an application are scheduled to run on different machines at the same time. Co-scheduling techniques relay on the communication behaviour of the application to schedule the communicating jobs simultaneously.

- Resource reservation: In a resource reservation policy, a job is allowed to reserve required resources even before having the job entering the system so it can ensure resource availability.

Obviously, these additional features introduce non-trivial overheads to the scheduler. Therefore, many schedulers are designed to execute the main scheduling functions only, to keep the scheduler simple and light in weight. This kind of scheduler is known as a best-effort scheduler, which means that the scheduler always tries to make the best decision for each job before it starts running, but with minimum performance overhead. It is an optimistic strategy that assumes an ideal running environment. Hence, a job would most likely never need special care, such as resource reservation or co-scheduling before starting, nor pre-emption, migration or rescheduling after starting.

4.4.4.3 Scheduling Procedure

Scheduling procedure refers to the scheduling algorithm that implements the scheduling policy. Two classes of scheduling algorithms can be addressed: optimum and sub-optimum algorithms, as shown in Figure 4.13.

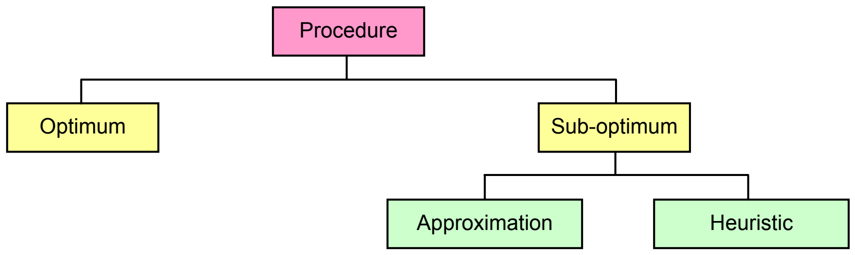


Figure 4.13: Scheduler Procedure Model

4.4.4.3.1 Optimum versus Sub-Optimum Algorithms

As explained in section 4.2, the scheduling problem belongs to a broad class of optimisation problems which has been subject to extensive research for decades. To solve optimisation problems, optimisation algorithms are constructed which try to find optimal solutions for which a certain objective function is at its optimum, i.e. less than or greater than a threshold value [29].

However, polynomial time optimisation algorithms cannot be constructed for all optimisation problems. These problems are considered to be NP-hard. In such cases, one often uses sub-optimal algorithms which tend towards, but do not guarantee, the finding of an optimal solution for any instance of the optimisation problem. Sub-optimal solutions are further divided, based on the approach followed to construct them, into approximation and heuristics algorithms, which are usually confused in the literature.

4.4.4.3.2 Approximation versus Heuristic

An approximation algorithm uses the same formal computational model used by an optimum algorithm, but instead of searching the entire solution space for an optimum solution, the algorithm is satisfied when a “good” solution is found. This technique is used to decrease the time taken to find an acceptable solution (schedule). In the case of heuristics, empirical data analysis is used to look for a “good” solution. A heuristic is a collection of rules or steps that guide one toward a solution that may or may not be optimal. Examples include greedy algorithms, Tabu search and simulated annealing [40].

Among distinguishing features between approximation algorithms and heuristics are the performance guarantee and evaluation. An approximation algorithm usually has a theoretical performance guarantee; for instance the solution it calculates is ten percent worse than the best solution. On the other hand, a heuristic will usually have no performance guarantee but its solution is intuitively close to the best solution [41].

4.5 Grid Resource Scheduling

As defined in Chapter 2, a grid is a collection of computational resources that are coupled together to solve a single large problem that cannot be solved on any single one of these resources. Hence, a specialised resource management system is usually employed to mitigate the complexity of managing such a large number of distrusted heterogeneous resources.

Generally, three basic functions are carried out by a grid resource management system:

- Recourse discovery.
- Allocating jobs to resources.
- Job and resource monitoring [42].

Although grid resource management and grid resource scheduling are used interchangeably among many grid practitioners, the second function, which is about allocating jobs to resources, is what is particularly meant by resource scheduling, as defined in section 4.2. This section highlights the main characteristics of grid schedulers and applies the proposed framework to compare three well established grid schedulers.

Due to the special characteristics of grid environments, as described in section 4.5.1, the grid resource scheduling problem is considered more demanding than other scheduling problems. Nonetheless, current work in grid scheduling involves many manual administrative works. Therefore, new research on grid scheduling should mainly focus on solving three problems:

1. Finding a good schedule.
2. Automating the scheduling process.
3. Building a flexible, scalable, and efficient scheduling mechanism [42].

More about the current state of the grid resource scheduling problem, and its certain nature and performance measures, are discussed in [43-45].

4.5.1 Characteristics of Current Grid Schedulers

The scheduling problem, in general, has been extensively studied in many areas and there is no clear evidence that grid scheduling is a new problem which is different from traditional scheduling [43]. However, grid scheduling is more challenging due to the special characteristics of grid environments and the current implementation of grid resource schedulers, which are summarised in following sections.

4.5.1.1 Centralised and Hierarchical Schedulers

Many current grid systems employ centralised schedulers to simplify the resource management process and insure full control over resources. There are also hierarchical schedulers at several different layers with a grid scheduler (meta-scheduler) at the highest level, a local scheduler (cluster scheduler) at the lowest level, and other layers may exist in between. Both schemes are based on the assumption that a detailed system state is available to schedulers which is highly expensive, considerably restricts the scalability of the system and simply unrealistic in many grid environments due to their dynamic nature.

4.5.1.2 Static Clairvoyant Schedulers

As indicated in section 1.1, virtually all current grid systems employ clairvoyant scheduling policies assuming prior availability of information about incoming jobs, such as execution times and release dates. Additionally, static schedules are usually generated in advance which is apparently unrealistic in dynamic environments and severely restricts the system flexibility.

4.5.1.3 Lack of Dedicated Access to Resources

Most grid resources are shared among several users or are available to grid usage only during idle cycles, dramatically affecting the predictability of resource availability [34]. It is important to note that in grid computing, the term dedicated resource is employed with a different meaning to that mentioned in section 4.4.1.1. Grid computing applications are sometimes run in background mode or as a screen saver only when the system is idle. In this case, it is said that the resource is not dedicated which means that it is not exclusively devoted for grid utilisation. A dedicated resource usually receives jobs from a single scheduler in contrast to non-dedicated resources, which receive workloads from multiple schedulers.

4.5.1.4 Heterogeneous Resources

The heterogeneous nature of grid resources results in great variation and unpredictability in the capability of resources. Based on resource models presented in section 4.4.1, the most common resource model that grid schedulers need to deal with is parallel unrelated resources with different processing speeds for each kind of job. There is also the parallel uniform resource model where resources vary in their processing speeds but the speed of each resource is constant for all type of jobs which is usually the case in Cluster Grids.

4.5.1.5 High Communication Latency

Until today, most grid environments have exhibited high communication latency [34]. Therefore, it is always believed that coarse-grained applications and independent jobs are better candidates to run on grids than applications that need intensive communication and synchronisation such as workflows.

4.5.2 Examples of Grid Schedulers

Historically the most common grid scheduler is the user [34]. Nowadays, many efforts are under way to change this situation. Condor [46], Legion [47] and Nimrod-G [48] among others, are dedicated schedulers utilised in grid resource management systems to assign jobs to machines. An extensive survey and taxonomy of grid scheduling systems is presented in [8-13]. Here we present a brief overview of one of the well-known schedulers in each performance metric. The intention is not to make a complete listing of grid schedulers, but to extract evident features of one well known example of each performance metric and apply the proposed framework, as shown in Table 4.1.

Table 4.1: Scheduling Framework Applied to Condor, Legion and Nimrod-G

	Features					Scheduler			Notes
						Condor	Legion	Nimrod/G	
Resource model	Parallel	Identical							
		Uniform				✓	✓	✓	
		Unrelated				✓	✓	✓	
	Dedicated	Flow shop							
		Open shop							
		Job shop							
Job model	Independent	BoT				✓		✓	
		DL				✓			
	Dependent	DAG					✓		Can support other models
		Non-DAG					✓		
Per. met.	Job centric					✓			
	Resource centric						✓		Can support other models
	Economy-based							✓	
Scheduler model	Organis.	Centralised				✓			
		Decentralised					✓	✓	
		Distributed	Cooperative						
			Non-cooperative			✓	✓	✓	
		Non-distributed							
	Pro.	Optimum							
		Sub-optimum	Approximation						
			Heuristic			✓	✓	✓	
	Policy	Stochastic							
		Deterministic	Non-clairvoyant						
			Clairvoyant	Static					
				Dynamic	Batch	✓	✓	✓	
					Immediate				

4.5.2.1 A Resource-Centric Scheduler: Condor

Condor [46, 49] is a resource management system for High Throughput Computing (HTC) environments, where the main goal is maximising the throughput. It was developed by University of Wisconsin, Madison in 1988. It leverages large collections of heterogeneous distributed computing resources, ranging from super computers to desktops, to solve independent coarse-grained computer-intensive jobs.

Condor implements a centralised distributed non-cooperative scheduling policy where a central node allocates loads to available nodes, then each node schedules the running of its own jobs. The “matchmaking” mechanism is used to match jobs to resources based on classified advertisements. The underlying scheduling algorithm works in batch mode and is based on the OSH where a job is assigned to the first idle machine. The end objective is to balance the load between machines. Load balancing is a heuristic that is based on the assumption that being fair to machines results in better system performance.

Condor has been successfully implemented in widely distributed computational grids, as demonstrated by SETI@home project [50]. Condor uses the pre-emption technique to stop grid jobs, giving the resource owner higher priority while using his own resources.

Condor-G is a new version of Condor, developed by University of Wisconsin, Madison in 2001. Condor-G leverages the advantages of both Condor and Globus ToolKit [51], the de facto standard for open source grid computing. Condor-G is designed to run more fine-grained jobs than Condor, and is more tolerant to faults.

4.5.2.2 A Job Centric Scheduler: Legion

Legion [47, 52] was developed by the University of Virginia in 1998. It is an object-oriented resource management system for High Performance Computing (HPC) where the main goal is to minimise the execution time of an individual job. Although the performance metric is resource-centric, this might be altered using application level schedulers such as Nimrod/G [48] and AppLeS [53]. This is because Legion provides several default generic schedulers, but it also allows users to enter their own application level schedulers. This has the advantage of allowing diverse job models to benefit from Legion as each application can have its own scheduler associated with it. The scheduling policy is decentralised with the scheduling decision made in periodic or batch modes. For better QoS, Legion allows resource reservation and rescheduling through job monitoring.

4.5.2.3 An Economy-Based Scheduler: Nimrod/G

Nimrod/G [48, 54] was developed by Monash University, Australia, in 2000 based on the Nimrod system. The Nimrod system has been utilised successfully in static scheduling but it is unsuitable for dynamic environments such as grids. Therefore, Nimrod/G has been developed to overcome this shortcoming. Nimrod/G is an economy-based resource broker. It focuses on applying economic theories to grid resource management and scheduling as part of the GRACE (Grid Architecture for Computational Economy) framework. Job models considered in Nimrod/G are parameter sweep applications where a job consists of one program with a large set of independent parameters to be studied. The program specifies the deadline and the price to pay for executing the program. Nimrod/G uses GRACE services to dynamically trade with resource providers and consumers. The scheduling policy is decentralised with the scheduling decision made periodically. For better QoS, Nimrod/G allows resource reservation.

4.6 Conclusion

Although the resource scheduling problem is a mature research area, a significant lack is a generic framework that can be applied to different application domains, and a unified taxonomy to cope with the different terminologies and inconsistency among technical terms. It is hoped that the work presented in this chapter succeeded in conveying a high level framework for previously published resource scheduling taxonomies, and clarifying areas of ambiguity and conflicts. The aim was to make a step forward to plug this gap.

Some scheduler features are usually used interchangeably in the literature ignoring the differences between them, for instance, dynamic versus adaptive, dynamic versus non-clairvoyant, decentralised versus distributed, and distributed versus cooperative. The differences between these features are presented in Table 4.2.

However, as resource scheduling systems are closely related to specific system and application models, it is difficult to complete a comprehensive survey of the overall spectrum. Therefore, this chapter has emphasised grid resource scheduling systems in particular in applying the proposed analysis framework.

Based on the different models of resource schedulers, addressed in this chapter, and the special characteristics of PM-Grids, highlighted in Chapter 3, the following features can be identified as the main features required for an efficient PM-Grid resource scheduler:

- Self-scheduling and cooperative to conceal the resource management complexity from the personal user.
- Decentralised, local and adaptive to cope with the highly dynamic environment.
- Non-clairvoyant to handle the unpredictability of incoming jobs.

Finally, it is important to note that there is a significant research potential for the non-clairvoyant scheduling which has not been previously applied in the context of grid resource scheduling and management systems.

Table 4.2: Differences between Interchangeably used Scheduler Features

	Features	Difference
versus	Decentralised	Authority for making policy decisions is distributed to multiple entities.
	Distributed	Responsibility for making and carrying out policy decisions is shared among multiple entities.
versus	Dynamic	Job information is available to the scheduler before it starts running.
	Non-clairvoyant	Job information is only available after the job finishes its execution.
versus	Dynamic	Parts of jobs information are revealed dynamically, thus schedules are generated in the same manner.
	Adaptive	Some decisions and parameters of the scheduling algorithm are dependent on the current system context.
versus	Approximation	Uses the same formal computational model of an optimum algorithm, but instead of searching the entire solution space for an optimal solution, the algorithm is satisfied when a "good" solution is found. It has theoretical performance guarantee.
	Heuristic	Uses empirical data analysis to look for a "good" solution that may or may not be optimal. It has no theoretical performance guarantee.

4.7 References

[1] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, "Taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," in *Proc. IEEE Symp. Reliable Distrib. Syst.,* pp. 330-335, 1998.

[2] J. Cao, A. T. S. Chan, Y. Sun, S. K. Das and M. Guo, "A taxonomy of application scheduling tools for high performance cluster computing," *Cluster Comput.,* vol. 9, pp. 355-371, 2006.

[3] T. L. Casavant, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Softw. Eng.,* vol. 14, pp. 141, 1988.

[4] H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems," *IEE Comput. Digital Tech.*, vol. 141, pp. 1-10. 1994.

[5] M. Dessouky, N. Morary and B. Kijowski, "Taxonomy of scheduling systems as a basis for the study of strategic behavior," *Human Factors,* vol. 37, pp. 443-472, 1995.

[6] J. J. Kanet and V. Sridharan, "Scheduling with inserted idle time: problem taxonomy and literature review," *Operational Research,* vol. 48, pp. 99-110, 2000.

[7] D. Quadt and H. Kuhn, "A taxonomy of flexible flow line scheduling procedures," *European J. Operational Research,* vol. 178, pp. 686-698, 2007.

[8] F. Dong, "A taxonomy of task scheduling algorithms in the Grid," *Parallel Process. Lett.s,* vol. 17, pp. 439-454, 2007.

[9] K. Krauter, R. Buyya and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Practice Experience,* vol. 32, pp. 135-164, February. 2002.

[10] S. Venugopal, R. Buyya and K. Ramamohanarao, "A taxonomy of Data Grids for distributed data sharing, management, and processing," *ACM Comput. Survey,* vol. 38, pp. 3, 2006.

[11] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *J. Grid Comput.,* vol. 3, pp. 171- 200, 2005.

[12] M. Wieczorek, A. Hoheisel, and R. Prodan, "Taxonomies of the multi-criteria grid workflow scheduling problem," in *Grid Middleware and Services*, US: Springer US, 2008, pp. 237-264.

[13] C. Jiang, C. Wang, X. Liu and Y. Zhao, "A survey of job scheduling in grids," *Lecture Notes Comput. Sci.*, vol. 4505. Berlin: Springer, pp. 419-427, 2007.

[14] M. R. Garey and D. S. Johnson. *A Guide to the Theory of NP-Completeness*. W. H. Freeman: San Francisco, 1979.

[15] S. Homer and A. L. Selman, *Computability and Complexity Theory*, New York: Springer, 2001, pp.194.

[16] J. Leung, L. Kelly and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* Boca Raton, FL: CRC Press, Inc, 2004,

[17] D. P. Spooner, J. Cao, S. A. Jarvis, L. He and G. R. Nudd, "Performance-Aware Workflow Management for Grid Computing," *Comput. J.,* vol. 48, pp. 347-357, 2005.

[18] A. Legrand, A. Su and F. Vivien, "Off-line scheduling of divisible requests on an heterogeneous collection of databanks," in *Proc. 14th Heterogeneous Comput. Workshop,* 2005, pp. 123a.

[19] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. A. B. Silva, C. O. Barros, C. Silveira and C. Silveira, "Running bag-of-tasks applications on computational grids: The MyGrid approach," in *Proc. Int. Conf. Parallel Process.,* 2003, pp. 407-416.

[20] G. C. Fox and D. Gannon, "Special Issue: Workflow in Grid Systems," *Concurrency and Computation: Practice and Experience,* vol. 18, pp. 1009-1019, 2006.

[21] W. Depoorter, R. Van den Bossche, K. Vanmechelen and J. Broeckhove, "Evaluating the divisible load assumption in the context of economic grid scheduling with deadline-based QoS guarantees," in *Proc. 9th IEEE/ACM CCGRID,* 2009, pp. 452-459.

[22] H. J. Kim and V. Mani, "Divisible load scheduling in single-level tree networks: Optimal sequencing and arrangement in the nonblocking mode of communication," *Comput. Math. with Appl.*, vol. 46, pp. 1611-1623, 2003.

[23] F. Dasilva and H. Senger, "Improving scalability of Bag-of-Tasks applications running on master–slave platforms," *Parallel Comput.,* vol. 35, pp. 57-71, Feb. 2009.

[24] V. Bharadwaj, *Scheduling Divisible loads in Parallel and Distributed Systems*, Los Alamitos, California: IEEE Computer Society Press, 1996.

[25] O. Beaumont, "Scheduling divisible loads on star and tree networks: results and open problems, " *IEEE Trans. Parallel Distrib. Syst.,* vol. 16, pp. 207, 2005.

[26] A. Kejariwal, A. Nicolau and C. D. Polychronopoulos, "History-aware Self-Scheduling," in *Proc. ICPP 2006,* pp. 185-192.

[27] R. Buyya, D. Abramson, J. Giddy and H. Stockinger, "Economic models for resource management and scheduling in Grid computing," *Concurrency and Computation: Practice and Experience,* vol. 14, pp. 1507-1542, 2002.

[28] D. J. Wilkinson, *Stochastic Modelling for Systems Biology.* Boca Raton: Taylor & Francis, 2006.

[29] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Eds. *Handbook on Scheduling: Models and Methods for Advanced Planning*

(International Handbooks on Information Systems). Secaucus, NJ: Springer-Verlag, 2007.

[30] M. Walker, "A Framework for Effective Scheduling of Data-Parallel Applications in Grid Systems," M.S. thesis, School of Engineering and Applied Science, University of Virginia, 2001.

[31] R. Motwani, S. Phillips and E. Torng, "Non-Clairvoyant Scheduling," *Theoretical Comput. Sci.,* vol. 130, pp. 17-47, 1994.

[32] Y. N. Sotskov, V. S. Tanaev and F. Werner, "Stability radius of an optimal schedule: A survey and recent developments," in *Industrial Applications of Combinatorial Optimisation.* G. Yu, Ed. Boston, MA: Kluwer Academic Press, 1998, pp. 72-108.

[33] I. K. Savvas and M. Kechadi, "Dynamic task scheduling in computing cluster environments," in *Proce. Int. Symp. Parallel Distrib.Comput.*, 2004, pp. 372-379.

[34] J. Nabrzyski, J. M. Schopf and J. Wceglarz, *Grid Resource Management. State of the Art and Future Trends* (International Series in Operations Research & Management Science). Boston, MA: Kluwer Academic Publishers, 2004.

[35] P. Tang, P. Yew and C. Zhu, "Impact of self-scheduling order on performance on multiprocessor systems," in *ICS,* 1988, pp. 593-603.

[36] C. Hsu and C. Carothers, "A self-scheduling model using agent-base, peer-to-peer negotiation, and open common schema," in *Proc. 17th Int. Conf. Production Research, 2003.*

[37] C. D. Polychronopoulos and D. J. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. Comput.,* vol. 36, no. 12, pp. 1425-1439, 1987.

[38] T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 87-98, 1993.

[39] C. Yang, K. Cheng, and K. Li, "On development of an efficient parallel loop self-scheduling for grid computing environments," *Parallel Comput.*, vol. 33, pp. 467-487, 2007.

[40] S. I. Gass, C. M. Harris, *Encyclopaedia of Operations Research and Management Science*, Secaucus, NJ: Springer-Verlag, Inc, 2007.

[41] Y. Xu, D. Xu and Jie Liang, *Computational Methods for Protein Structure Prediction and Modeling.* New York: Springer, 2007.

[42] J. Joseph and C. Fellenstein, G*rid Computing*, N.J.: Prentice Hall Professional Technical Reference, 2004, pp. 378.

[43] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj, D. Snelling, "Open issues in grid scheduling," UK e-science Tech. Rep., 2004.

[44] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure (the Morgan Kaufmann Series in Computer Architecture*

and Design). Morgan Kaufmann, 2003,

[45] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *Proc. HPDC-11 2002*, pp. 352-358.

[46] Condor Project [online]. Available: http://www.cs.wisc.edu/condor, [accessed Feb. 2, 2010].

[47] Legion: A Worldwide Virtual Computer [online]. Available: http://legion.virginia.edu/, [accessed Feb. 2, 2010].

[48] DSTC Nimrod/G [online]. Available: http://www.csse.monash.edu/~sgaric/nimrod/, [accessed Feb. 2, 2010].

[49] M. Litzkow, M. Livny and M. Mutka, "Condor - A hunter of idle workstations," in *Proc. 8th Int. Conf. Distrib. Comput. Syst.,* Jun. 1988.

[50] SETI@home [online]. Available: http://setiathome.ssi.berkeley.edu/, [accessed Feb. 2, 2010].

[51] Globus Alliance [online]. Available: http:// www.globus.org, [accessed Feb. 2, 2010].

[52] A. S. Grimshaw and A. Natrajan, "Legion: Lessons learned building a grid operating system," in *Proc. IEEE,* vol. 93, 2005, pp. 589-603.

[53] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su and D. Zagorodnov, "Adaptive computing on the grid using AppLeS," *IEEE Trans. Parallel Distrib. Syst.,* vol. 14, pp. 369-382, Apr. 2003.

[54] R. Buyya, D. Abramson and J. Giddy, "Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid,"*, in Proc. 4th Int. Conf./Exhibition High Performance Comput. Asia-Pacific Region,* 2000, pp. 283-289.

Chapter 5
HoPe: A Honeybee Inspired Scheduler

5.1 Introduction

The key to any successful grid system is an efficient scheduler that allocates available resources to incoming jobs. Indeed, measuring the potential and usefulness of a grid system is nothing but exploiting its ability to efficiently schedule its underlying resources [1]. The extremely dynamic nature, diversity and limited capabilities of resources, as well as difficulties in predicting the nature and timing of incoming jobs, are all factors considerably scaling the complexity of the scheduling problem in PM-Grids.

Through observation, the honeybee colony faces an extraordinarily difficult scheduling problem in nature, while allocating forager bees to nectar sources during the Nectar Acquisition Process (NAP). The honeybee colony efficiently solves this problem through simple non-intelligent agents (honeybees), running a decentralised cooperative and adaptive self-scheduling policy. This observation motivated the research for this thesis to follow a heuristic approach for resource scheduling in PM-Grids that mimics the techniques followed by honeybees during the NAP.

Among the main contributions of this chapter are the introduction of HoPe: a Honeybee inspired resource scheduling heuristic for Personal Mobile Grids as well as a detailed analysis of the NAP from a resource scheduling perspective.

Section 5.2 defines the scheduling problem in PM-Grids. Section 5.3 identifies the main non-functional requirements of HoPe. In section 5.4 the HoPe broad hypothesis is stated and the questions it raises are addressed and answered. Section 5.5 explains and analyses in depth the NAP in honeybee colonies, and builds an abstract model for the process, pointing to the main features that inspired HoPe. Section 5.6 identifies HoPe implementation elements and explains the analogy of the PM-Grid to honeybees, as well as the analogy of HoPe to the NAP. Section 5.7 presents a brief review of other biologically inspired scheduling heuristics, comparing them to HoPe while section 5.8 concludes the chapter.

5.2 Scheduling Problem in PM-Grids

A PM-Grid is a unified collection of resources connected via a PN. It has the potential to deliver grid level services to a personal end user. Whenever a need occurs, a PM-Grid user uses his/her client device to send a computational job for execution on his/her PM-Grid. The job is received at the nearest work-space. Hence, a variable unpredictable stream of incoming jobs arrives at each work-space from client devices. Executer devices need to be efficiently allocated to incoming job streams producing results that are sent to the result-space where an unpredictable stream of generated results arrives. All results that belong to one job are accumulated in a separate output file. When an output file containing all job results is ready, it is allocated to a composer device for final preparation before being dispatched to the sender or a requested address.

As in the case of all grid systems, the core of a PM-Grid is a scheduler which strives to efficiently assign tasks to available grid resources. Grid resource scheduling is a complex problem in general, as detailed in section 4.5. Centralised plan-ahead schedulers are usually deployed for this purpose. In these schedulers, a single authority is in charge of all decisions regarding who should run what and when, as shown in Figure 5.1. Two assumptions are common in such schedulers: First, clear and sufficient information about incoming jobs is known in advance, which is simply not realistic. Second, a globally detailed and frequently updated view of the system resources state is available [2], which is prohibitively expensive, severely restricts the scalability of the system and exposes it to high security risks.

As indicated in section 4.4.4.2.2, assuming the availability of clear information about the incoming jobs before making the scheduling decision, is what is referred to as clairvoyant scheduling, with which virtually all grid resources are concerned. Although this clairvoyant assumption considerably simplifies the scheduling problem, it is not valid for most real world problems [3]. In contrast, the non-clairvoyant scheduling approach assumes that such information is unavailable in advance, making it more practical for many computer engineering problems, especially grid computing where it is usually difficult and costly to make reasonable predictions.

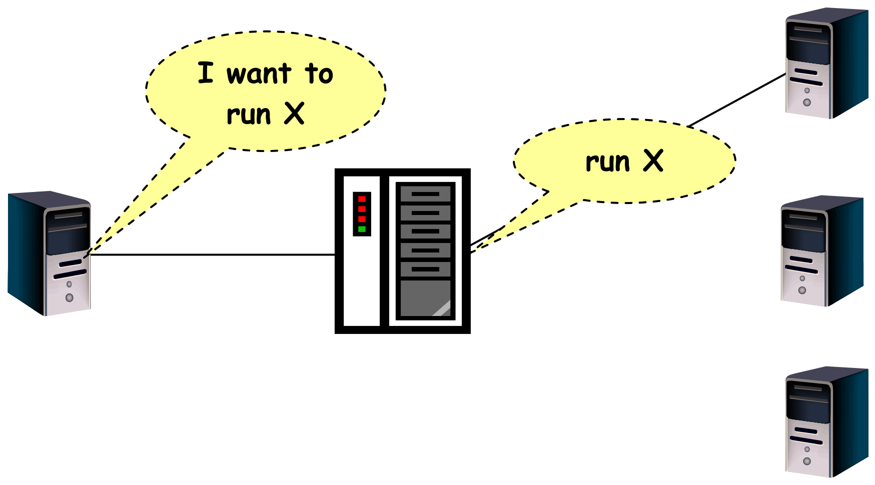


Figure 5.1: Conventional Grid Schedulers

In a nut shell, the scheduling problem in a PM-Grid can be defined as efficient non-clairvoyant scheduling in a highly dynamic environment of limited resources. The non-clairvoyant scheduling problem is considered as NP-hard as it contains two classical NP-hard problems as special cases:

- The first case, when all tasks are sequential, the problem reduces to the multiprocessor scheduling problem which is NP-hard [4].
- The second case, when all tasks have the same execution time, the scheduling problem becomes the bin-packing problem which is NP-hard also [5].

Therefore, one practical way to solve this problem is to design a heuristic that tries to find a "good" solution for this extraordinarily difficult scheduling problem [6].

5.3 HoPe Requirements

The nature of the scheduling problem in PM-Grids, as described in the previous section, suggests that an efficient scheduler should meet the following non-functional requirements:

- The scheduling policy should be a self-controlled decentralised scheme to hide the management complexity of underlying resources from the personal user and to cope with the highly dynamic environment.

- The scheduler should follow an adaptive non-clairvoyant policy to cope with the unpredictability of incoming jobs and the high variability in available resources.
- The processing complexity and the time needed to make the scheduling decision must be maintained at minimum levels to keep PM-Grid agents simple enough to fit in mobile devices.
- Agents must communicate in a reliable scheme, again due to the highly dynamic nature of the system where devices can leave and join, or switch on and off, at any time.
- The communication between agents should be minimised to reduce the power consumption of mobile devices and also to cope with the dynamic environment.
- Agents should have cooperative non-competitive behaviour as underlying resources are usually owned by one person who sets common system goals which all devices need to jointly accomplish.

5.4 Broad Hypothesis

Based on the nature of the scheduling problem and requirements of resource schedulers in PM-Grids, applying traditional grid scheduling schemes, as illustrated in Figure 5.1, searching for an optimal algorithm is simply not feasible. Therefore, this thesis, proposes a novel approach to grid scheduling with a non-clairvoyant, fully distributed and adaptive self-scheduling scheme, as shown in Figure 5.2. This scheduling approach aims specifically to address the complexity of the scheduling problem in PM-Grids which is dramatically scaled by the high level of dynamism, diversity and limited capabilities of underlying resources, as well as the remarkable unpredictability of the nature and timing of incoming jobs.

Observations of honeybees has revealed that the colony faces an extraordinarily difficult scheduling problem in nature, due to weather unpredictability and food variability, while allocating honeybees to nectar sources during the Nectar Acquisition Process (NAP). The honeybee colony efficiently solves this problem through simple non-intelligent agents, (honeybees) running a decentralised cooperative and adaptive self-scheduling policy. The aim is to maximise the nectar intake while maintaining the hive at a stable state where nectar collecting and honey processing rates are balanced. This observation is the foundation of the broad hypothesis behind HoPe:

Efficient non-clairvoyant scheduling in a highly dynamic environment of limited resources may be achieved with a heuristic approach based on simple agents. The agents allocate themselves to multiple work sources in a decentralised, cooperative and adaptive self-scheduling scheme striving to maximise work intake while maintaining the system in a stable state, in an attempt to imitate the behaviour of honeybees during the NAP.

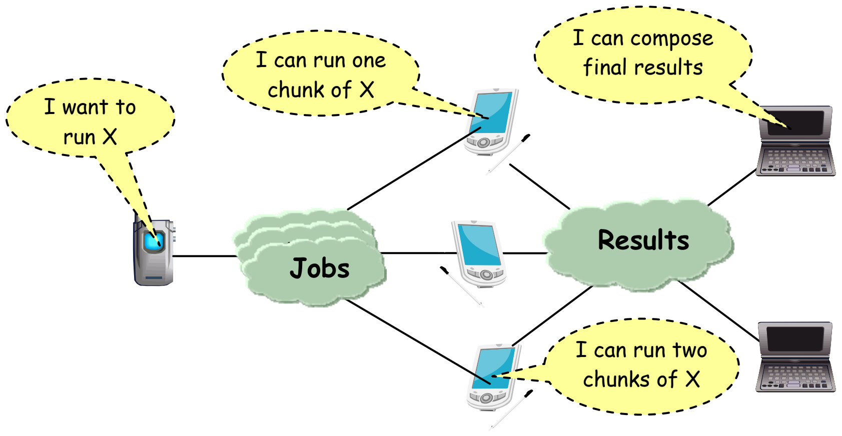


Figure 5.2: HoPe Scheduling Approach

Three obvious questions regarding the HoPe broad hypothesis might arise:

1. Why a heuristic approach?
2. Why the honeybee colony based inspiration?
3. Why stability as a scheduling objective?

5.4.1 Why a Heuristic Approach?

In many complex scheduling problems, it is more efficient to have a heuristic suggesting a “good” (near optimal) schedule rather than evaluating all possible schedules. For instance, consider developing a schedule for 30 different jobs (or a single job of 30 internal tasks) and five different machines. In this case, a scheduler needs to examine 5^{30} possible mappings of jobs to machines before determining the optimum schedule. Assuming that the scheduler consumes only one nanosecond to evaluate the quality of one possible schedule, then the scheduler will need 5^{30} nanoseconds ($> 4 \times 10^{10}$ sec. > 1000 years) to evaluate all possible schedules [7].

What makes heuristics more efficient for many scheduling problems are not only that they are considerably easier and quicker to develop than optimisation algorithms, but most importantly heuristics are generally more robust to changes in data as well. This is because a heuristic deliberately but judiciously ignores certain computationally expensive data and depends mainly on alternative simpler parameters. Indeed, designing an efficient heuristic is mainly about knowing exactly what information to ignore and what information to retain.

Ideally, expensive parameters to gather, maintain and manipulate are ignored. This ignorance frees the scheduler from burdens of reading and manipulating such information. Hence, the decision produced is independent of the ignored information and unaffected by their changes [8]. On the other hand, alternative simpler parameters, which are easier to gather, maintain and manipulate, are retained. These simple parameters are usually correlated to the system performance in an indirect instead of direct way. They have an impact on the efficiency of the provided schedule but may not be directly related, in a quantitative way, to the system performance [3].

Hence, in this thesis it is expected that taking a heuristic approach will produce an efficient scheduler with “good” system performance. However, as the case with all heuristics, this thesis does not aim to prove that there is a first-order relationship between the heuristic proposed and the desired results.

5.4.2 Why Honeybee Colony-based Inspiration?

A heuristic based on an intelligent agent approach may considerably reduce the complexity of a scheduling problem. However, the processing complexity and communication cost of launching intelligent agents are usually overwhelming and significantly consume resources especially in devices of limited capabilities. Therefore, this thesis has followed the basic idea behind the swarm intelligence approach, where intelligent behaviour emerges from the interaction of simple non-intelligent agents.

Swarm intelligence is based on the fact that social insects (insects that live in colonies) such as bees, ants and termites, present an intelligent collective behaviour albeit composed of simple individuals of limited capabilities [9]. This intelligent collective behaviour emerges naturally from the special characteristics of these simple agents which include:

- Self-organisation: Unsupervised coordination of activities.
- Adaptiveness: Response to highly dynamic environments.
- Robustness: Accomplishing the group objective even if some members of the group malfunction.

These properties lend themselves well to distributed optimisation problems in telecommunications, manufacturing and transportation, among others [10, 11]. The rationale behind the swarm intelligence approach is apparent. Social insect colonies are efficient successful paradigms from nature and following the same principles of such systems will produce successful counterpart engineering solutions.

Basically, the process of designing a swarm intelligence, or more generally a "bio-inspired" [12] solution, can be summarised by the flowchart presented in Figure 5.3 which includes the following steps:

1. Define the engineering problem at an abstract level.
2. Find a biological system with the same abstract problem.
3. Build an abstract model for the biological system.
4. Build an abstract model for the engineering problem to mimic the abstract biological model.
5. If possible, increase the similarities at a finer level of detail.
6. Test and evaluate.
7. If results are not acceptable, go to step 2.

It is clear that finding the right biological system is the most critical step in this process. In this thesis, the honeybee colony has been chosen because the problem of allocating resource limited machines to job sources in a highly dynamic environment has an apparent correspondence with the problem of allocating forager bees to nectar sources in the virtually unpredictable conditions of weather changes and food availability in nature, as explained in section 5.4. Indeed, as stated in [13]:

> Among all social systems, the social physiology underlying the food collection process of honeybee colonies might be the greatest metaphor of cooperative group functioning outside the realm of human society.

However, utilising ideas from honeybees has not been explored in grid computing, to the best of our knowledge. Therefore HoPe has been introduced as a step forward to plug this gap, exploring the potential of honeybee based algorithms in mitigating the grid level resource management complexity.

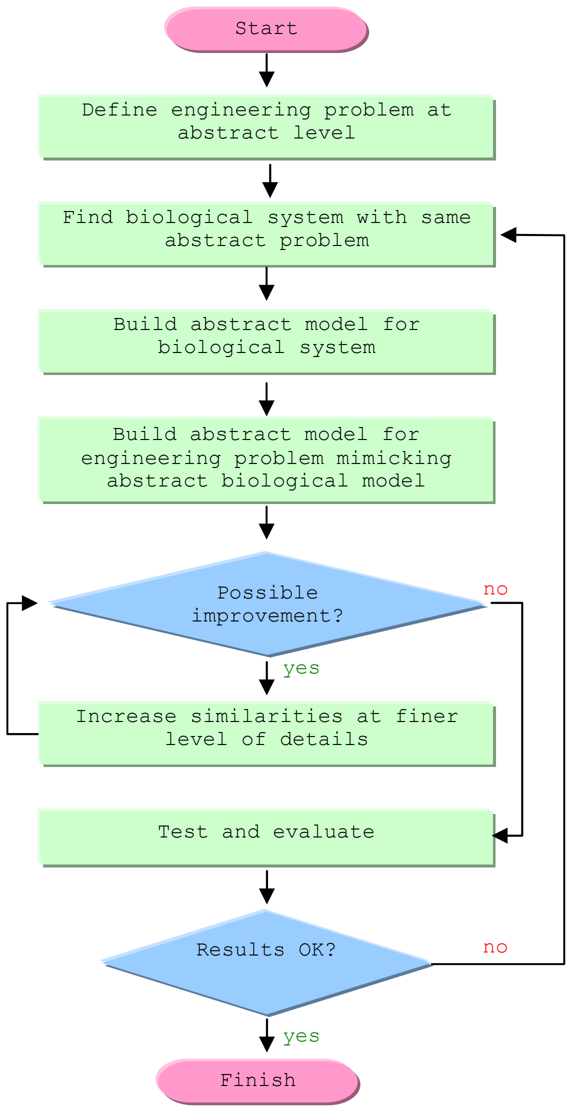

Start
Define engineering problem at abstract level
Find biological system with same abstract problem
Build abstract model for biological system
Build abstract model for engineering problem mimicking abstract biological model
Possible improvement?
no
yes
Increase similarities at finer level of details
Test and evaluate
Results OK?
no
yes
Finish

Figure 5.3: Bio-inspired Design Process

5.4.3 Why Stability as a Scheduling Objective?

Basically, there are two performance metrics that are commonly used in evaluating grid resource scheduling systems: turnaround time (TT) and throughput.

The TT, also known as response time and completion time, measures the elapsed time from when a client submits a job until the client receives the corresponding results. It is the most popular metric in computational grids. This measure indicates the system performance which relates to how well scheduled resources are being used to the benefit of each user of the system.

On the other hand, the throughput represents the amount of work completed by the system over a period of time, or per time unit. The throughput is the main concern of all high throughput computing systems. This measure relates to how efficient the system is in regard to the added cost or overhead associated with the resource scheduler. It indicates how well system resources are being used to the benefit of all users of the system. However, maximising the throughput on its own saturates the network and deteriorates other performance aspects such as queuing delay, which in turn affects the TT. Therefore, a mechanism is needed to control the rate of job injection into the system [14].

Indeed it is always difficult to compromise between scheduling performance measures, as maximising the throughput may come at the expense of TT, while minimising TT might come at the expense of throughput. The simultaneous evaluation of both measures, throughput and TT, is very difficult as they represent conflicting goals [15]. Therefore, a methodology is required where these measures are separately observable [1] and/or new performance measures that help to optimise both are required for capturing the tradeoffs [16].

Consequently, this thesis focuses on the stability performance measure where the objective is to maximise the job collection rate subject to minimising the difference between job collection and result generation rates. Stability controls the rate of job injection into the system. Hence, it is critical for bounding the queue size which presumably reflects positively in TT and throughput. However, proving a first-order relationship between the stability measure on one side, and TT and throughput on the other, is considered beyond the aim of this thesis.

Stability measure is of extra importance for this study in particular, due to the suspected bottleneck in the workstation with the hive queue, as explained in section 5.5.2. In bottleneck cases, scheduling decisions should focus on the bottleneck resource, in an effort to maximise its production rate and work output from that point, and trying not to release work faster than the bottleneck can process, in order to maintain the stability of the resource [17].

5.5 The Nectar Acquisition Process (NAP)

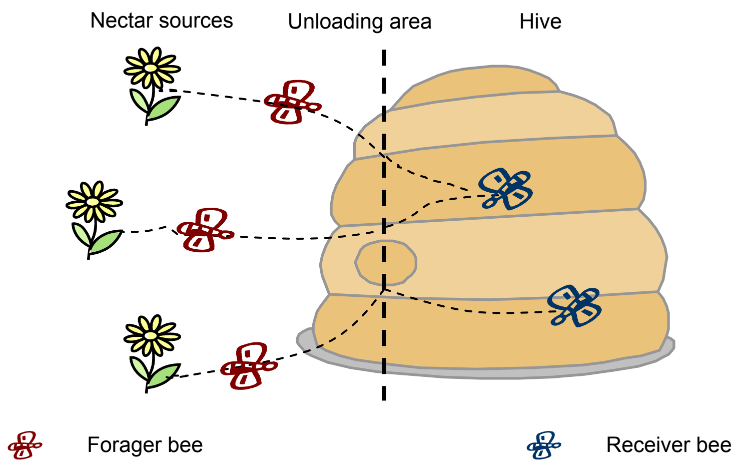


Figure 5.4: Nectar Acquisition Process (NAP)

Since ancient times, scientists have been fascinated with the social organisation of the honeybee colony. This has been translated into many studies of its biology, such as [13, 18-20] among others. As the main aim of this study is to show the possibility of exploiting the honeybee food collection technique in designing a solution for a particular computer engineering problem, only selected background, that is necessary to understand the basic idea behind this work, is provide based on [13].

A honeybee colony has a limited number of bees which it needs to allocate wisely to the surrounding flower patches from which they collect nectar and bring it to the hive for further processing in order to generate comb honey. This process, illustrated in Figure 5.4, is what has been referred to as the Nectar Acquisition Process (NAP).

During NAP, a honeybee colony divides labour, based on temporary specialisation, among two groups: forager bees, who work in fields collecting nectar from food sources turning it into raw honey, and receiver bees, who work in the hive processing raw honey to produce comb honey (honey-filled wax comb as stored directly by the bees). This organisation boosts the efficiency of the NAP, but requires dynamic coordination of the two labour groups to keep the rates of nectar collection and honey processing in balance.

This coordination problem is significant because the colony experiences large and unpredictable variations in the nectar availability. The colony adjusts its nectar collection and honey processing rates with respect to external nectar supply mainly by dynamically adjusting the number of forager and receiver bees through "waggle" and "tremble" dances.

When food sources are laden with nectar, the colony increases the number of forager bees, raising the nectar collection rate. This is done through the waggle dance which stimulates some receiver bees to change their roles to foragers and help in nectar foraging.

When the processing rate is lowered, having a number of receiver bees changed their role to forager bees, the colony speeds up the honey processing rate through tremble dance. The tremble dance stimulates some forager bees to work as receiver bees, as shown in Figure 5.5. In the following sections we present a detailed analysis for the NAP from the resource scheduling perspective.

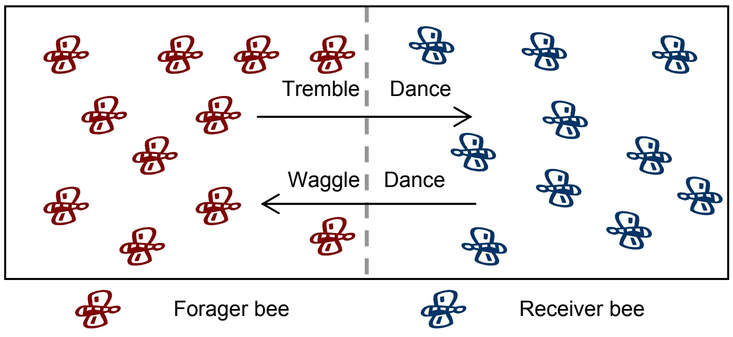


Figure 5.5: Dynamic Reallocation of Labours during NAP

5.5.1 Abstract Algorithms

The main steps followed by a forger bee and a receiver bee are described in section 5.5.1.1 and section 5.5.1.2 respectively.

5.5.1.1 Forager Bee Abstract Algorithm

Initially, a forager bee starts searching randomly for a nectar source. When a nectar source is found, the forager sucks an amount of nectar that fits its stomach where it is mixed with proteins and enzymes producing raw honey. Before returning to the hive, the forager assesses the profitability of the remaining nectar in the food source.

On return, the forager waits in the unloading area, an area near the entrance of the hive, for a receiver bee to unload her honey. The forager assesses the waiting time (*WT*) in relation to its tremble dance threshold (*TDT*), which is an internal variable calculated dynamically by each forager based on its experience. “Long” *WT* means that the colony nectar collection rate is markedly high. Thus, a nectar forager experiences a long *WT* because most receiver bees are busy unloading already arrived foragers. In response, the forager bee performs a tremble dance, in which the bee walks slowly about the nest making trembling movements to boost the number of receiver bees. The duration of this dance is closely correlated with the *WT* experienced by the dancing bee.

If the *WT* is “not that long” but the profitability of the nectar in the food source, from where the forager bee gathered nectar, is “high” when compared to the waggle dance threshold (*WDT*), which is an internal variable calculated dynamically by each forager based on its experience, the bee starts a waggle dance. During the waggle dance, the bee flies in a small figure-of-eight on the dancing floor, a small area inside the hive. The direction and duration of this dance is closely correlated with the direction and profitability of the nectar in the food source being advertised by the dancing bee. The aim of this dance is to boost the number of forager bees targeting this food source and also to recruit idle receiver bees to work in nectar collection in order to increase the colony’s nectar collection rate.

After having its raw honey unloaded, a forager bee needs to decide where to look next for nectar. If there is still some nectar in her last visited source, it flies directly there, otherwise it needs to check the dancing floor for any waggle dance. If there is, the forager selects a dancer randomly to follow, then flies directly to the advertised nectar source. If this is not the case, the forager looks around for any tremble dancing bee, if found, the forager bee changes its role to a receiver bee working in honey processing. If none of the above is the case, the forager bee starts searching randomly again for a nectar source.

Hence basically, a forager bee searching procedure can be summarised by the high level algorithm presented in Figure 5.6, while the full high level algorithm that shows the basic steps followed by a forager bee can be summarised informally by the flowchart and pseudo code presented in Figure 5.7 and Figure 5.8 respectively.

```
If you already know a good source
        Fly directly to that source.
Else if you do not know but a friend knows a good source
        Fly to that source.
Else if neither you nor any of your friends know a good source
        Search randomly.
```

Figure 5.6: Basic Idea of Local Search in NAP

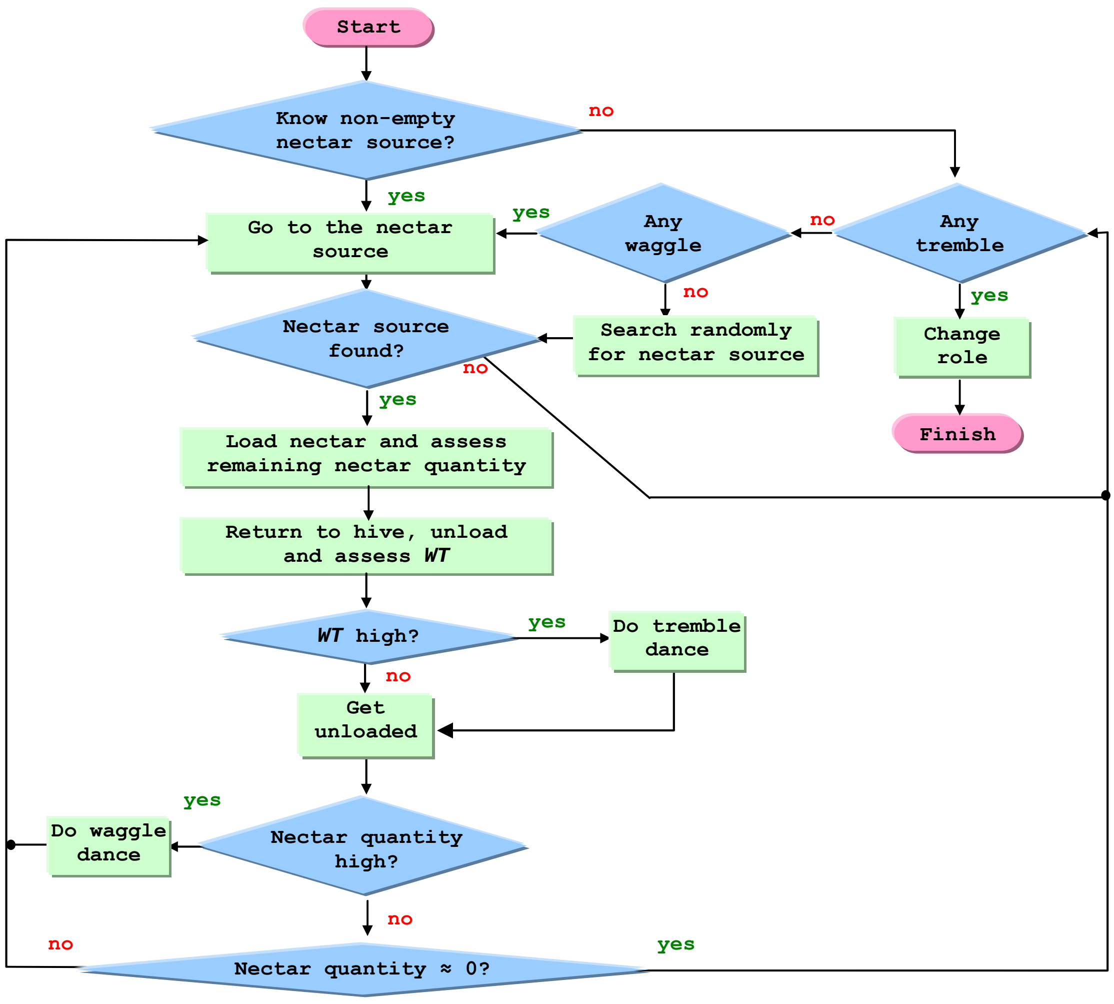


Figure 5.7: Forager Bee High Level Flowchart

```
1. Each forager bee
2. Loop for ever
3.    If you already know a non-empty nectar source
4.      Fly to that source
5.      If not available anymore or there is no more nectar in the source
6.          Go to step # 22
7.      Load nectar
8.      Assess quantity of remaining nectar ($NQ$)
9.      Generate raw honey
10.     Return back to hive
11.     loop
12.         If there is no ready receiver bee in the unloading area
13.            Assess $WT$ since you arrived back hive
14.            If ($WT \geq TDT$)
15.                Do tremble dance
16.     until a receiver bee arrives
17.     Let the receiver bee unload your raw honey
18.     If ($WT \leq TDT$) and ($NQ \geq WDT$)
19.         Do waggle dance
20.   End if
21.   Else
22.     If you can see any tremble dancer around
23.         Change your role to receiver bee
24.         Exit
25.   End else if
26.   Else
27.     If there is any waggle dancer bee in the dancing floor
28.         Choose one waggle dancer to follow randomly
29.         Go to step # 4
30.   End else if
31.   Else
32.     If there is not any waggle dancer bee in the dancing floor
33.         Search randomly for a nectar source
34.         If you find any
35.               Go to step # 7
36.   End else if
37.End Loop
```

Figure 5.8: Forager Bee High Level Pseudo Code

5.5.1.2 Receiver Bee Abstract Algorithm

Initially, a receiver bee is waiting near the unloading area for the arrival of any returning forager bee loaded with raw honey. Once one arrives, the receiver bee unloads its raw honey to store in a comb, a hexagonal cell made of bee wax. When a comb is full, the receiver bee fans its wings to thicken the honey and cap it with wax producing comb honey.

During her waiting time, a receiver bee also keeps an eye on the dancing floor. If it detects any waggle dancer, it changes its role to a forager bee flying to the advertised nectar source. The abstract algorithm of the basic steps followed by a receiver bee can be summarised informally by the flowchart and pseudo code presented in Figure 5.9 and Figure 5.10 respectively.

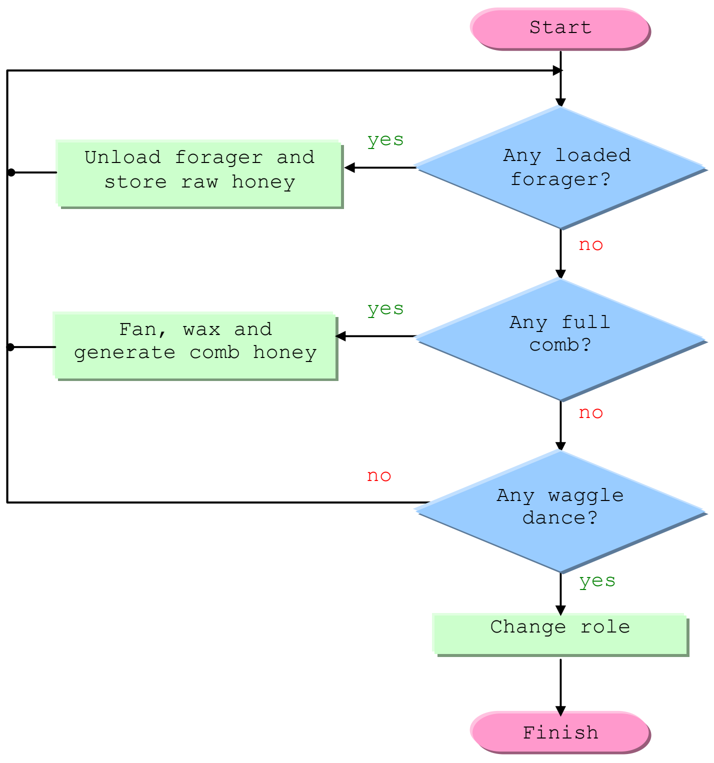


Figure 5.9: Receiver Bee High Level Flowchart

```
Each receiver bee
Loop for ever
    If any loaded forager bee arrives hive
        Unload her and store raw honey into a comb
    Else if you can see any full comb
        Fan and wax the comb generating comb honey
    Else if there is any waggle dance
        Change your role to forager bee
        Exit
    End else if
End Loop
```

Figure 5.10: Receiver Bee High Level Pseudo Code

5.5.2 Abstract Queuing Model

Principally, to build a system model, the system is simplified as much as possible by including only the main properties and functions while eliminating finer details that complicate matters. Generally system models are classified as:

- Mathematical models (Analytical models): A mathematical model is an abstraction of the real system represented as a set of equations summarising the aggregate system performance but does not describe the detailed events that occur in the real system.

- Simulation models (empirical or experimental models): A simulation model is an experiment that mimics events that occur in the real system, allowing experimentation with different parameters and control logic.

Some attempts to mathematically model the foraging behaviour of honeybees have already been published. In [21] a differential equation of dynamic labour allocation in honeybees has been proposed and evaluated for one set of experimental conditions. A generic nonlinear differential mathematical model for social foraging in both ants and bees has been suggested in [22]. In [23] a probabilistic model of individual-level sensing, decision making and nectar foraging in honeybees has been developed. Additional detailed models that attempt to quantify most features of honeybees are presented in [24].

However, these are concrete mathematical and probabilistic models quantifying features of the honeybee foraging behaviour based on certain sets of predefined assumptions. The problem with this approach is that the honeybee colony, as in the case of all biological systems, has unique characteristics that are apparently different from the mathematical assumptions that lie beneath analytical models. For instance, the honeybee colony employs an adaptive control strategy based on the current system state while analytical models usually evaluate steady state conditions only. An analytical model measures the system behaviour using expected values for a predefined set of performance metrics ignoring any changes in the system behaviour over time [17].

Therefore, in modelling the NAP, this thesis has initiated a queuing theory based approach. A generic model for the NAP is developed as a queuing network then this model is simulated in several representative scenarios. Detailed descriptions of simulation scenarios are presented in Chapter 6.

The queuing theory is used to model and analyse systems that involve waiting for services. A queuing system model usually includes one or more pools (queues) of arriving elements and one or more servers (processors) attached to the pools.

Based on this, from the queuing theory point of view the NAP, or more generally a honeybee colony, includes the following components:

- Queues: There are two groups of queues in the NAP:
  - Nectar source queues: consist of S small parallel queues of nectar waiting for forager bee processors to serve them.
  - The hive queue: is a large queue of gathered nectar (raw honey), waiting for receiver bee processors to serve them producing comb honey.
- Processors: There are $N = N_c + N_p$ processors in the NAP. They are organised in two main groups :
  - Forager bee processors: consist of $N_c = \sum_{i=1}^{s} N_i$ processors that are assigned to the nectar source queues. They represent forager bees collecting nectar from nectar sources.
  - Receiver bee processors: consist of N_p processors that are assigned to the hive queue. They represent receiver bees engaged in unloading and processing raw honey.

Components of a honeybee colony are organised, for the NAP, as a two stage open queuing network (in open networks, arriving items can join and leave the system, whereas in closed networks the total number of items within the system remains fixed), as illustrated in Figure 5.11. This queuing network is composed of multiple workstations for the nectar collection course (first stage) and one workstation for the honey processing course (second stage). Each workstation is composed of an input queue and one or more servers. Processed items from all collection course workstations are placed in a single output queue (hive) which in turns serves as an input for the single processing course workstation.

Interestingly, the design choice of the network of workstations, Figure 5.11, underlying this natural system, the honeybee colony, is more efficient than other alternative design choices such as parallel or serial servers shown in Figure 5.12. Both models of networks of workstations and parallel servers are generally preferred over the serial servers model. However, in highly variable environments, a single group of multiple servers, corresponding to a network of workstations, is more efficient than parallel servers, each with its own queue [17]. An important feature of this model is that it is highly dynamic. The number of processors (N_c and

N_p), arrival rates (λ_1, λ_2,..., λ_s, λ_p) and also the number of queues *S,* are variable over time.

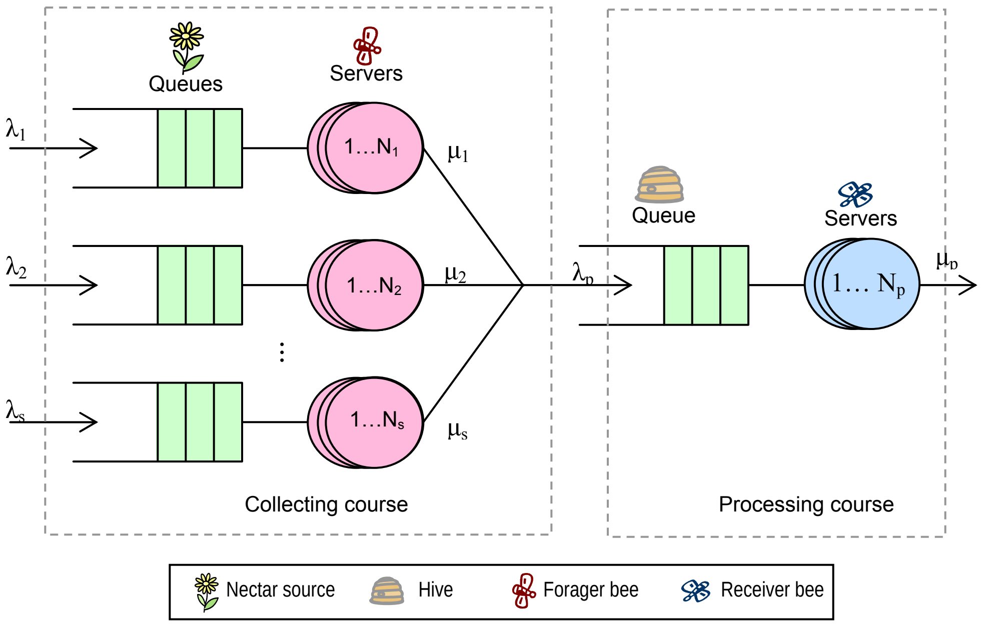


Figure 5.11: Honeybee Colony Queuing Model

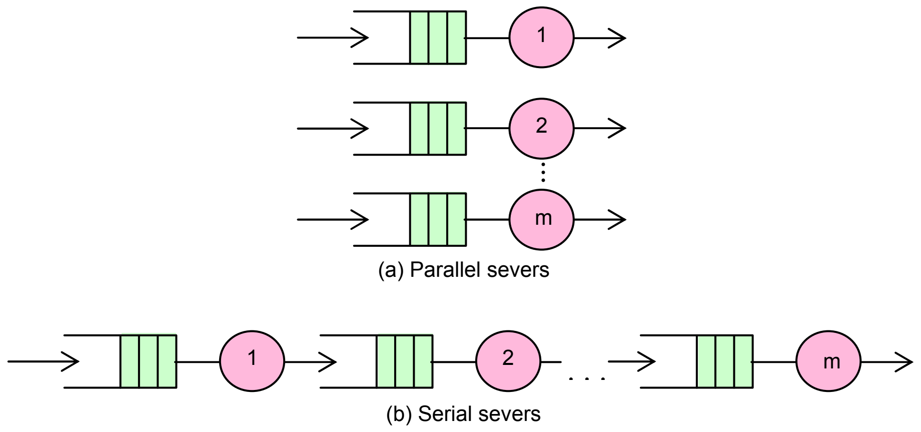


Figure 5.12: Alternative Queuing Models to a Network of Workstations

Based on the above-mentioned features of the NAP model, it is clear that it is difficult to fit the NAP scheduling problem under the mathematical scope of the classical queuing theory. Therefore, computer-based simulations are used to approach the problem.

However, the single processing course workstation can be a "bottleneck" in this system. Therefore, it is critical for the entire system to maintain a constant flow, which is known as the steady state balance equation [17]:

$$\textit{Rate out} = \textit{Rate in} \tag{5.1}$$

Hence, the scheduling objective of the whole system has been chosen carefully to enhance system stability, as explained in section 5.3.3.

5.5.3 Formulation of the NAP Scheduling Problem

As illustrated by the queuing model of the NAP in Figure 5.10, the NAP can be divided into two stages: nectar collection course and honey processing course.

5.5.3.1 Nectar Collection Course

A number of N_c forager processors are connected to multiple nectar queues. A forager processor N_i assigns itself an average volume of L_c nectar load, from a nectar queue, based on its capacity. N_i delivers its load to a single hive queue, for further processing. N_i spends an average time of T_c to complete a collection course. The collection course is defined as the process from when N_i starts the decision as to which nectar source to access in order to load nectar, until it delivers its load to the hive queue, starting the decision making process again. The objective of the colony system during this cycle is to maximise its nectar collection rate (λ_p) which is a function of three variables [13]:

$$\lambda_p = N_c L_c / T_c \tag{5.2}$$
$$T_c > 0,$$

where:
N_c is the number of forager bees engaged in nectar collection
L_c is the average volume of nectar load per forager
T_c is the average time of a collection cycle.

However, strong evidence suggests that the principal means the system uses to adjust λ_p is altering N_c rather than L_c or T_c [13]. Hence (5.2) can be rewritten as:

$$\lambda_p = aN_c \tag{5.3}$$
$$a > 0$$

Hence, the scheduling problem in the collection course of the NAP can be defined as: How to allocate a set of N_c parallel processors to S sources of divisible load jobs, so that the number of delivered jobs per time unit is maximised:

$$\text{Maximise } \{F(N_c) = aN_c\} \qquad (5.4)$$
$$a > 0$$

5.5.3.2 Honey Processing Course

A number of N_p receiver processors are connected to a single hive queue. A receiver processor N_j takes an average volume of L_p honey load, processes it to generate comb honey as necessary. N_j spends an average time of T_p to complete a processing course. The processing course is defined as the process from when N_j receives a honey load until it processes it and is ready to receive another honey load. The objective of the colony system during this cycle is to maximise its honey processing rate (μ_p) which is a function of three variables [13]:

$$\mu_p = N_p L_p / T_p \qquad (5.5)$$
$$T_p > 0,$$

where:

N_p number of receiver bees engaged in honey processing

L_c average volume of honey load per receiver bee

T_p average time of a processing cycle.

Strong evidence suggests that the principal means the system uses to adjust μ_p is changing N_p rather than L_p or T_p [13]. Hence (5.5) can be rewritten as:

$$\mu_p = bN_p \qquad (5.6)$$
$$b > 0$$

The scheduling problem in the processing course of the NAP can now be defined as: How to allocate a set of N_p parallel processors to a set of P jobs, so that the number of delivered jobs per time unit is maximised:

$$\text{Maximise } \{F(N_p) = \mu_p = bN_p\} \qquad (5.7)$$
$$b > 0$$

Based on the two above-mentioned courses, the end objective of the scheduling problem in the NAP can be seen as maximising both nectar collection and honey processing rates:

$$\textit{Maximise}\ \{F(N_p)\,,\ F(N_c)\ \} \qquad (5.8)$$

Subject to minimising the difference between them:

$$\textit{Minimise}\ \{|\ F(N_p) - F(N_c)\ |\} \qquad \textit{(5.9)}$$

As outlined in [13]: "the rates of nectar collecting and processing must be kept in balance for the overall operation to proceed. If the collecting rate exceeds the processing rate foragers will experience long unloading delays upon return to the hive. Reciprocally, if the processing rate exceeds the collecting rate, nectar receivers will be underemployed".

5.5.4 Main Features

After analysing the social foraging behaviour of honeybees during the NAP, six main features can be identified as the main drivers of this thesis inspiration:

1. Decentralised self-control scheduling policy.
2. Adaptive non-clairvoyant scheduling policy.
3. Easily calculated local control variables.
4. Reliable communication technique.
5. Economic communication scheme.
6. Non-competitive cooperative behaviour.

Noticeably, these features have a direct correspondence to the requirements of PM-Grid resource schedulers as addressed in section 5.3.

5.5.4.1 Decentralised Self-Control Policy

During the entire NAP, a honeybee colony shows a complete absence of any form of central or hierarchical control. There are no certain authorities giving instructions to other bees regarding who should do what and when; rather each honeybee makes these decisions for herself independently of all other bees.

5.5.4.2 Non-Clairvoyant Adaptive Scheduling Policy

The scheduling scheme followed in the NAP is based on non-clairvoyant scheduling policy; it does not require or depend on any information about incoming work. It is also highly adaptive. This can be clearly exemplified by the dynamic allocation of labour among worker bees. Many social insects exhibit a division of labour among their members which features them in controlling complex systems [25]. However, these labours are permanent such as in ant and termite workers [26]. Within a honeybee colony, forager and receiver bees, dynamically exchange their roles based on the supply (nectar collection) and demand (honey processing) as shown in Figure 5.5. This temporary specialisation makes the system more robust and flexible under different loads and scales. Adaptability in honeybees can also be exemplified by the way dancing thresholds are determined by each bee. Each dancer individually decides a dancing threshold for itself based on its perception of the current system state.

5.5.4.3 Easily Calculated Local Control Variables

As indicated in section 5.5.3, the objective of a honeybee colony is to maximise its nectar collection and honey processing rates, while maintaining them in balance. Forager bees perform waggle and tremble dances for this purpose. Surprisingly, a forager bee starts dancing without knowing the values of these two important global variables (nectar collection rate and honey processing rate). Instead, it monitors two local variables which are correlated with the global variables but are far easier to calculate: the waiting time experienced until a receiver bee arrives and the profitability of the last visited nectar source. Relying on local non-expensive parameters makes the scheduler lighter in weight in terms of implementation and more robust in dynamic environments.

5.5.4.4 Reliable Communication Scheme

In social insects, such as ants, communication among colony mates usually takes place in any location in or out of the shared environment. Shared information, especially outside the shared environment, can be easily altered by external elements. This usually has a negative impact on the reliability of shared information. In a honeybee colony, important information related to the NAP is exchanged only in a centralised shared memory inside the hive, where dances are performed. Neither colony mates nor external elements can alter the shared information. This not only ensures high reliability of exchanged information but also makes it much easier to exchange.

5.5.4.5 Economic Communication Scheme

Social insects usually communicate important information with their colony mates through implicit messages that alter their shared environment by pheromones or odours. This communication scheme usually takes time before being effective in attracting attention. In contrast, honeybees communicate important information, such as the need for more workers in a certain labour and locations of profitable nectar sources, through explicit advertisements for such information. This has an immediate effect in attracting attention and results in a very efficient group recruitment scheme. Furthermore, there are only two pieces of information that are needed to be exchanged in the colony during the NAP: the location of rich sources, and the need for more workers in a certain labour group. Besides exchanging only a limited amount of information, this information is very small in size, and its frequency is remarkably low [13].

5.5.4.6 Non-Competitive Cooperative Behaviour

The main driver of many social groups' behaviour, including human beings, is to maximise their profit as defined in certain terms such as money or food. This is not the case in honeybees. A honeybee does not compete with other bees within its colony to get more profit in terms of food; it cooperates with them to reach common goals. This might clarify why honeybees are not choosy when exploiting food sources. A dance follower randomly chooses a dance to follow flying directly to the advertised source without waiting for the whole dance duration when the source profitability can be determined. Through this behaviour, foragers efficiently distribute themselves among all food sources. If instead, a forager bee tries to maximise its own benefit, it would have waited until it knows the profitability of a nectar source. This would have resulted in an all-or-none response which is a less than optimal allocating scheme [13].

5.5.5 Elements of Honeybee Colony and NAP

Modelling a process involves representing the environment where the process runs and activities and elements that are related to that process. Having a closer look at the NAP, one can differentiate between two groups of important elements: the first group represents elements that are part of the honeybee colony itself, which can be considered as the process environment. This group is used to model the PM-Grid. The second group represents elements and activities that constitute the NAP. This group is used to implement HoPe. Here, the two groups of elements are presented with a brief description of each.

It is important to note that this separation between elements of the honeybee colony as a system, and the NAP as a procedure to run by this system, is only to simplify tracing the origin of the PM-Grid and HoPe elements. In reality it is difficult to make such a distinction.

5.5.5.1 Elements of Honeybee Colony

Section 5.5.2 revealed the following important elements of a honeybee colony in the context of the NAP:

Agents:

- Forager bees: simple agents collecting nectar from food sources, transforming it into raw honey and bringing it to hive.
- Receiver bees: simple agents receiving raw honey from foragers and processing it further to produce comb honey.

Places:

- Food sources: multiple variable places from which nectar can be collected.
- Hive: a centralised well identified place where raw honey is delivered then packed for final processing as a comb honey.

5.5.5.2 Elements of NAP

Analysing the NAP identifies the following main elements:

Communication elements:

- Nectar: an input item that is collected and processed, producing honey.

- Raw honey (gathered nectar): an intermediate element produced after processing nectar. It is accumulated in uncapped combs (wax cells) for further processing.
- Comb honey: an output item that is produced after processing accumulated honey in full wax cells.

Communication means:

- Dancing floor: an area within the hive where important information regarding rich food sources is advertised through dancing.
- Unloading area: an area at the entrance of the hive where bees returning from food sources wait for food receiver bees to unload them.
- Combs: a place where raw honey is accumulated until full then transferred outside the hive.

Communication techniques:

- Waggle dance: a symbolic advertisement performed by a forager bee regarding the location of a rich nectar source and its profitability. It is also used to recruit idle receiver bees to work in nectar foraging to increase the nectar collection rate. The waggle dance is defined by three parameters: waggle dance threshold (*WDT*), waggle dance duration, advertised work-space.
- Tremble dance: a symbolic advertisement performed by a forager bee when it experiences a long delay waiting to be unloaded. The objective is to recruit idle forager bees to work as receiver bees to increase honey processing rate. The tremble dance is defined by two parameters: tremble dance threshold (*TDT*) and tremble dance duration.
- Forager waiting in the unloading area: an implicit request for unloading.
- Receiver waiting in the unloading area: an implicit message of being ready to unload.

Parameters:

- Waiting time: the time from which point a forager bee enters the unloading area, inside the hive, until a receiver bee arrives to unload her.

- Profitability: a measurable criterion of the quality of a nectar source. Much evidence suggests that it is related to the amount of nectar remaining in the nectar source.
- Nectar collection rate: the amount of gathered nectar (raw honey) arriving at the hive from all nectar sources per time unit.
- Honey processing rate: amount of raw honey processed inside the hive per time unit to produce comb honey.
- Waggle dance threshold (*WDT*): is an internal variable for each forager. It is a criterion related to the quantity of nectar remaining in the nectar source; when met a forager bee starts a waggle dance. It does not have a fixed value; instead it varies from bee to bee and from time to time under different conditions.
- Advertised nectar source: a specific nectar source with high profitability from which a waggle dancing forager bee has just returned.
- Waggle dance duration: the time from when a waggle dancing bee starts a waggle dance, until it finishes. It is a function of the profitability of the last visited nectar source.
- Tremble dance threshold (*TDT*): is an internal variable to each forager bee. It is a criterion related to the waiting time experienced by a forager bee on return from a nectar source; when met a forager bee starts tremble dance. It does not have a fixed value; instead it varies from bee to bee and from time to time under different conditions.
- Tremble dance duration: the time from when a tremble dancing forager bee starts a tremble dance, until it finishes. It is a function of the waiting time.

5.6 From Inspiration to Algorithm

The initial aim was to explore possibilities for defining the process of allocating machines to job sources, collecting tasks, processing them and generating results in a PM-Grid, in a way that is similar to the process of allocating forager bees to nectar sources, collecting nectar, processing it and generating comb honey in a honeybee colony. From the first glance, it was clear that the two problems were similar, at least at an abstract level. Therefore the aim has been extended to increase the similarity between the two systems at a finer level of detail. This section, briefly illustrates the direct correspondence between elements of the honeybee colony and the PM-Grid. After that, HoPe implementation elements are introduced, attempting to mimic elements from the NAP.

5.6.1 Mapping between PM-Grid and Honeybee Elements

As indicated in Chapter 3, a PM-Grid includes three groups of architectural elements: clients, workers (decomposers, executers and composers) and spaces (work-spaces and a result-space). A honeybee colony, as explained in section 5.5.5.1, includes two main groups of elements: agents (forager and receiver bees) and places (food sources and a hive). The direct correspondence between elements of the two systems is summarised in Table 5.1 and can be explained as follows:

- Spaces: Work-spaces and the result-space are analogues of important places to a honeybee colony, namely, food sources and the hive respectively.
- Workers: Executers and composers correspond to the two labour groups: forager and receiver bees respectively. Decomposers are needed to partition jobs into smaller tasks to fit executer capacities. In a honeybee colony, each forager sucks up a suitable amount of nectar based on her stomach size. Therefore, in the PM-Grid modelling, the decomposition functionality is integrated within the executers.
- Clients: Clients are basically the sources of jobs available in work-spaces; they are essential for PM-Grids to populate work-spaces with jobs. However, nectar is available in food sources naturally.

Table 5.1: Mapping between PM-Grid and Honeybee Colony Elements

Honeybee colony	PM-Grid
Food sources	Work-spaces
Hive	Result-space
Forager bees	Executers
Receiver bees	Composers

The design of a PM-Grid depicted in Figure 3.10 can be abstracted as a set of queues and servers as shown in Figure 5.13 which shows elements of PM-Grids corresponding to each queuing element at the top of it.

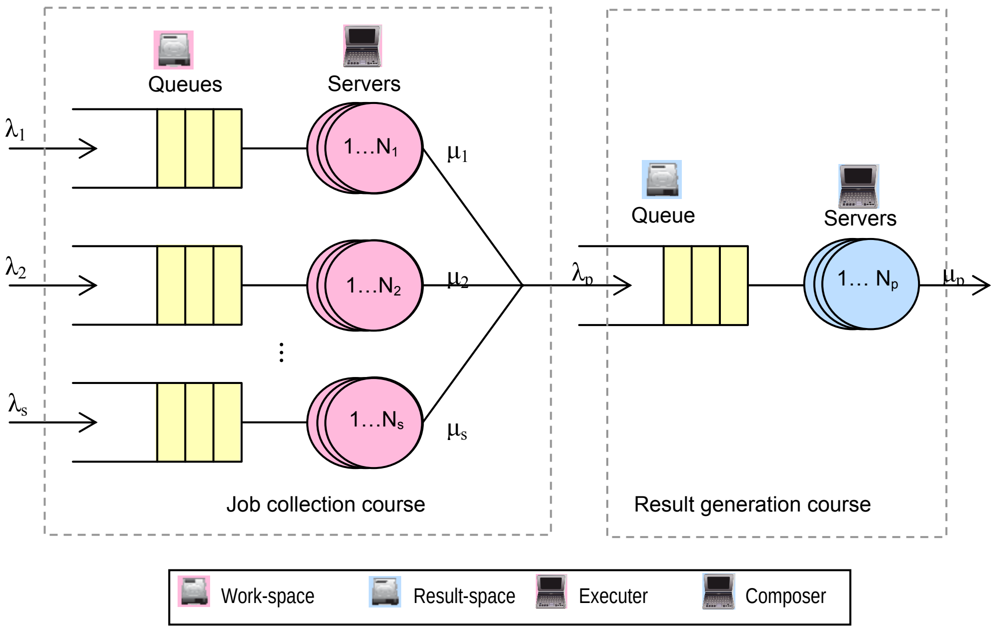


Figure 5.13: Queuing Model of a PM-Grid

5.6.2 HoPe Elements

HoPe design includes the following groups of elements:

Communication elements:

- Job: a large computational program that can be divided into an arbitrary number of smaller tasks.
- Task results: generated output after executing a task.
- Job results: generated output after composing task results from all tasks that belong to the same job.

Communication media:

- Executer help list (*EHL*): is a public dynamic list, residing in the result-space. It keeps track of the (*Executer Help Message*) *EHM* sent from executers to the result-space. It has entries for all active *EHM*s, which have not yet expired. For heavy loaded work-spaces, more *EHM*s will be received. As a result, they will have more entries in the *EHL* raising the probability of picking one of them randomly. Hence more executers will be attracted to heavy loaded work-spaces.

- Composer help flag (*CHF*): is a public structure, also residing in the result-space. It has two fields: the first field is a one bit flag which is set to one on receiving a *CHM* and unset to zero when the message duration is reached. The second field indicates the duration after which the flag should be unset. If a new *CHM* is received while the flag field has already been set to one, the duration field is updated with the duration value of the more recent *CHM*.
- Task results list: is a public structure, also residing in the result-space. Each entry to this list consists of two fields. The first contains the name of the file where received task results of a job are accumulated. The second field represents the status of this file. *Complete* status means that all task-results of this job are available and the file is ready for a composer. *Pending* status means the task-results have not been completed yet.

Communication techniques:

- Executer help message (*EHM*): a message sent by an executer to the result-space which includes the ID of a heavily loaded work-space and its profitability. It is used to attract more executers to this specific work-space, as well as to recruit idle composers to work as executers. Each message has two main fields: one indicating the duration after which the message expires and the other for the ID of the advertised work-space. The duration field is calculated as a function of the *RW* in the advertised work-space.
- Composer help message (*CHM*): a message sent by an executer to the result-space when it experiences a long time waiting for a *RM* from the result-space to indicate that the result-space can accept incoming task-results. The *CHM* has a main field for the duration after which the message expires. The message duration is calculated as a function of the *WT* experienced by the executer.
- Unload request message *(URM)*: a message sent by an executer to the result-space after generating task results checking if the result-space is ready to accept incoming results.

- Ready message (*RM*): a message sent by the result-space to an executer in response to an *URM* to indicate that the result-space is ready to accept task-results. For each task result picked by a composer, a *RM* is sent (or is ready to be sent) to respond to an *URM*.

Parameters:

- Remaining workload (*RW*): the volume of work, remaining in the last visited work-space.
- Waiting time (*WT*): the time from when an executer sent an *URM* to the result-space until it receives a *RM* from the result-space indicating that it can accept incoming task results.
- Job collection rate (*JCR*): is the number of jobs entering the result-space per time unit. It is calculated as:

$$JCR = TCR/k \qquad (5.10)$$
$$k > 0$$

  where:

  TCR is the task processing rate, the number of tasks entering the result-space per time unit.

  k is the average number of tasks per job.

- Result generating rate (*RGR*): is the number of jobs leaving the result-space, after having their job-results composed successfully, per time unit.
- Executer help threshold (*EHT*): is an internal variable to each executer that is related to how the executer assesses the *RW* in relation to its current workload (*CW*). When *EHT* exceeds a certain limit, the executer sends an *EHM* to the result-space. It does not have a fixed value; instead it varies from executer to executer and from time to time under different conditions. In HoPe implementation, an *EHM* is sent when the following condition is true:

$$RW > c \times CW \qquad (5.11)$$
$$c \geq 1,\ CW > 0$$

  where:

  c is an experimentation parameter.

CW is the current workload by the executer.

- Advertised work-space: The ID of a work-space of which the *EHT* is exceeded.
- Executer help message duration (*EHMD*): the elapsed time during which an *EHM* will be displayed in the *EHL*. In HoPe implementation the *EHMD* is calculated as a function of *RW* and *CW*:

$$EHMD = RW/CW \qquad (5.12)$$
$$CW > 0$$

- Composer help threshold (*CHT*): is an internal variable to each executer. It is a criterion related to the *WT* experienced by an executer waiting for a *RM* from the result-space. When *CHT* exceeds a certain limit, the executer sends a *CHM* to the result-space It does not have a fixed value; instead it varies from executer to executer and from time to time under different conditions. In HoPe implementation, a *CHM* is sent when the following condition becomes true:

$$WT > d \times ECD \qquad (5.13)$$
$$d \geq 1$$
$$ECD = Time\ URM\ sent - time\ job\ received \qquad (5.14)$$

where:
ECD is the execution cycle duration.
d is an experimentation parameter.

- Composer help message duration (*CHMD*): the elapsed time during which the *CHF* remains at one after being set by a *CHM*. It is calculated as a function of the *WT* experienced by the executer sending the *CHM*:

$$CHMD = e \times WT \qquad (5.15)$$
$$e \geq 1$$

where:
e is an experimentation parameter.

5.6.3 Mapping between NAP and HoPe Elements

There is a clear correspondence between the allocation problems of workers and machines. The aim is to increase the similarity at a finer level of detail through extensively borrowing the principles behind the NAP in the HoPe heuristic to tackle the resource scheduling problem in PM-Grids. The detailed mapping between NAP elements and HoPe elements is presented in Table 5.2.

Table 5.2: Mapping between NAP and HoPe Elements

NAP	HoPe
Nectar	Tasks
Raw honey	Task results
Comb honey	Job results
Dancing floor	Executer Help List
Unloading area	Composer Help Flag
Combs	Task results list
Waggle dance	Executer Help Message
Tremble dance	Composer Help Message
Forager waiting in the unloading area	Unload request message
Receiver waiting in the unloading area	Ready message
Profitability	Remaining workload
Waiting Time	Waiting Time
Nectar collecting rate	Job collecting rate
Honey processing rate	Result generating rate
Waggle dance threshold	Executer help threshold
Tremble dance threshold	Composer help threshold
Advertised nectar source	Advertised work-space
Waggle dance duration	Executer help message duration
Tremble dance duration	Composer help message duration

5.6.4 HoPe Algorithms

HoPe operates in two stages: an initialisation stage and a dynamic scheduling stage. In the initialisation stage, initial device roles are assigned. As indicated in Chapter 4, the result-space is co-located with the PN agent and is advertised by it at the PN formation stage which is described in details in the Technical Annex [27] and the conceptual PN architecture [28].

Work-spaces are identified and registered with the result-space in the current-work-spaces-list. Worker devices are also identified and registered with the result-space in the active-workers-list. The result-space updates both lists frequently through periodic Hello messages. Worker devices access the result-space when they need to update their copies of the current-work-spaces-list.

The dynamic scheduling stage of HoPe is presented as abstract algorithms for executers, as illustrated in Figure 5.14 and Figure 5.15, as well as composers as illustrated in Figure 5.16 and Figure 5.17. It is important to note that these algorithms are meant to serve as skeletons; implementation details such as data preparation, parameter passing, data structure, differ according to the requirements of various applications and running environments.

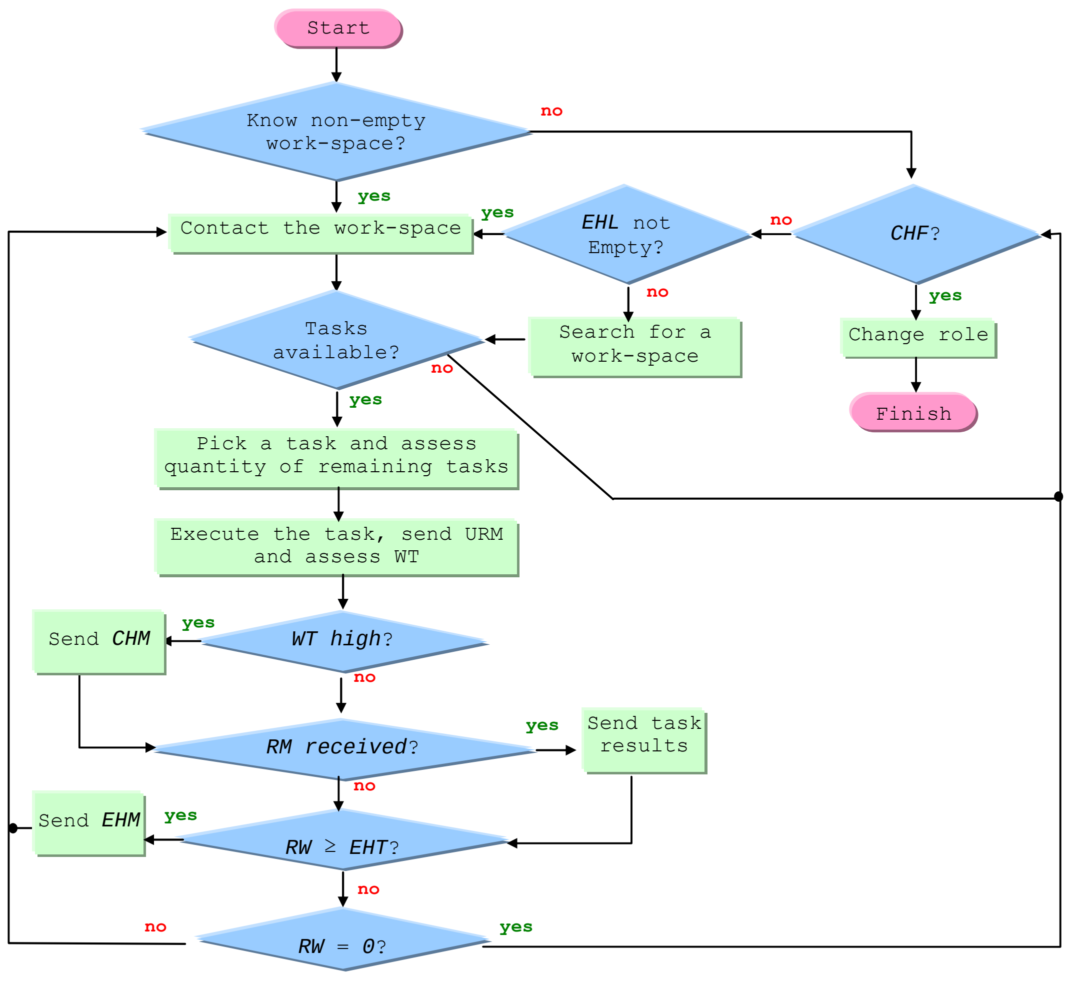


Figure 5.14: Executer High Level Flowchart

```
1.  Each executer
2.  Loop for ever
3.     If RW in last visited work-space >0
4.       Contact that work-space
5.       If not available anymore or there is no more workload in it
6.           Go to step # 24
7.       Get an amount of workload based on your capacity
8.       Calculate RW
9.       Execute the task
10.      Send URM to the result-space
11.      loop
12.          If you have not yet received RM from the result-space
13.          Calculate elapsed time from when you send URM
14.          If ($elapsed\ time \geq CHT$)
15.              Send CHM to the result-space
16.      until a RM received or URM time out
17.      If RM received
18.          Send task results to the result-space
19.          $WT = elapsed\ time$
20.      If ($WT \leq CHT$) and ($RW \geq EHT$)
21.          Send EHM to the result-space
22.    End if
23.    Else
24.      If CHF
25.          Change your role to composer
26.          Exit
27.    End else if
28.    Else
29.      If EHL is not empty
30.          Pick one message randomly
31.          Go to step # 4
32.    End else if
33.    Else
34.      If EHL is empty
35.          Select a work-space randomly form current-work-spaces-list
36.          Go to step # 4
37.    End else if
38. End Loop
```

Figure 5.15: Executer High Level Pseudo Code

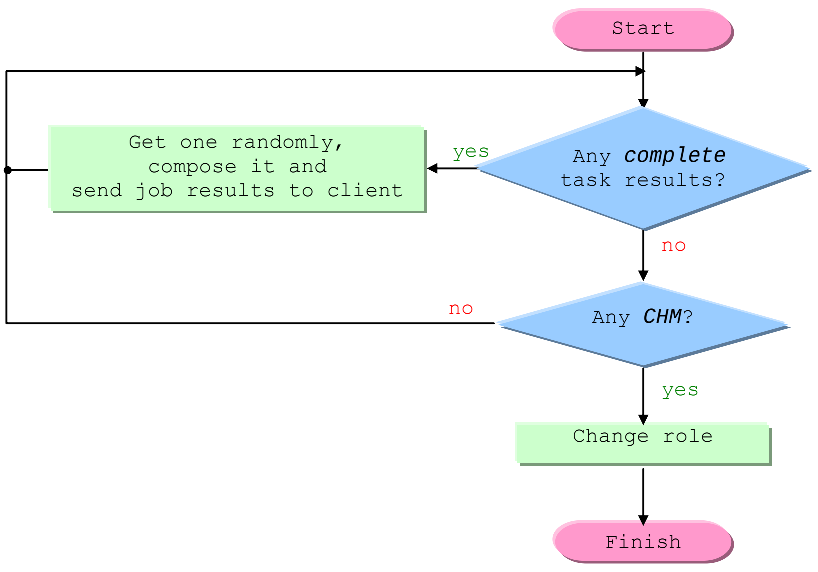

Start
Any complete task results?
yes
Get one randomly, compose it and send job results to client
no
Any CHM?
no
yes
Change role
Finish

Figure 5.16: Composer High Level Flowchart

```
Each composer
Loop for ever
    If there is any complete task results in the task results list
       Pick one randomly
       Compose it generating job results
       Send job results where client requested
    End if
    Else if EHL is not empty
       Change your role to executer
       Exit
    End else if
End Loop
```

Figure 5.17: Composer High Level Pseudo Code

A simple sequence diagram that shows the interactions between the main participants in HoPe during a job life cycle is illustrated in Figure 5.18. It is important to note that the aim of the sequence diagram is to show the main messages that are related to a single job life cycle, hence frequent messages and messages that are related to previous or incoming jobs are not shown in the diagram.

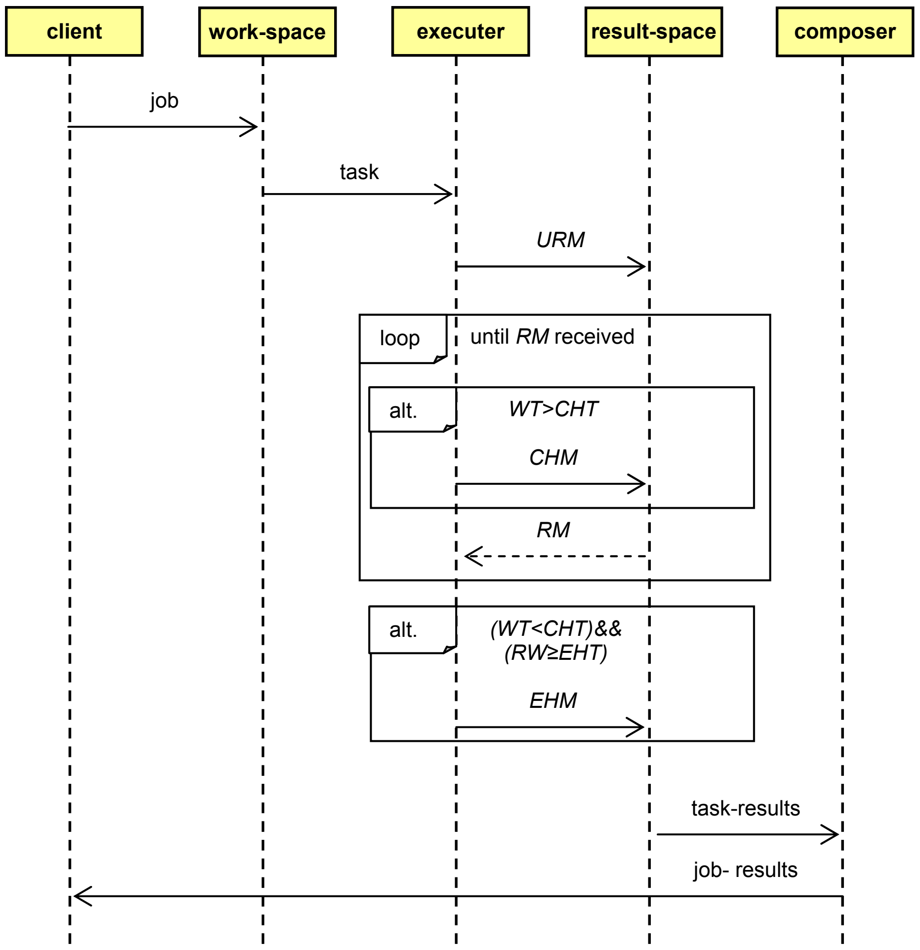


Figure 5.18: HoPe Sequence Diagram during a Job Life Cycle

5.7 Related Work

Resource scheduling is a very active area of research in general and in grid computing in particular. Chapter 4 extensively reviewed the area and pointed to some well-established grid schedulers such as Condor [29], Legion [30] and Nimrod-G [31]. This section focuses on insect inspired scheduling algorithms highlighting how they differ from HoPe.

Social insects are increasingly attracting attention in solving optimisation problems resulting in many successful new computing paradigms [32]. In [10] an overview of biological facts about social insects, their inspired algorithms and application areas in computer engineering and science, are presented.

Ant Colony Optimisation (ACO) [33], Particle Swarm Optimisation (PSO) [34] and more recently Artificial Bee Colony (ABC) [35] among others are well established meta-heuristics aiming to solve general optimisation problems. However, in many cases, it is difficult, or more efficient, to adapt the solution to a specific end problem. Indeed, there is a need for tailored algorithms to model certain problems as close to reality as possible. Hence, some work has already emerged to address this problem in resource scheduling.

Among all social insects, ant and bee colonies in particular have inspired researchers in resource scheduling. Consequently, it is timely to compare the basic ideas behind each algorithm. In section 5.5, the abstract algorithms for food (nectar) acquisition process in honeybees were presented. Here, the same process (food collection) is briefly described when performed by an ant colony.

For food collection, a group of worker ants start searching randomly for food sources. They leave a pheromone trial on the searching path while moving. When an ant discovers a source, it evaluates its profitability. During the return trip, the amount of pheromone an ant leaves on the path is proportional to the profitability of the discovered source. Other ants follow the path with a higher pheromone concentration [36].

There are already several publications that have proposed ant-inspired resource scheduling heuristics for grid environments [37]. For instance, in [38] an ant-inspired scheduling heuristic has been proposed for computational grids with the goal to minimise the execution time of computational jobs. Once a job is submitted to the grid, a worker will try to schedule this job to the node with the highest pheromone which is the node that gives the least job execution time for a test program. Although this algorithm gives promising results when evaluated, it suffers from some problems which are common to ant-inspired heuristics when compared to bee-inspired heuristics. First, information about a good source for ants is not directly advertised. It takes time until the pheromone reaches a certain concentration, then when an ant passes nearby or is guided by another ant, it will get to know about this source. On the other hand, in a bee colony a good food source is immediately advertised to the entire colony. Second, to make a decision, an ant needs to compare several alternatives which is time consuming and requires information about other alternatives from outside. This is not the case in bees where the decision is completely local to the bee itself requiring no outside information. Third, worker ants do not exchange their roles resulting in a fixed number of servers for each role, whereas worker bees are assigned their roles based on temporary specialisation which results in a more flexible system with a virtual number of servers.

In an investigation of whether pheromone-based algorithms (inspired by ants) are outperformed by non-pheromone based algorithms (inspired by bees), experimental results by [39] showed that non pheromone based algorithms are significantly faster when finding and collecting food and use fewer time steps to complete a task.

It seems that honeybees in particular behave in a very interesting way in achieving remarkable results, viewing them from the problem solving perspective. However, considering bees in the resource scheduling context remains relatively unexplored.

A very detailed exhaustive literature review and classification of the emerging studies in honeybee inspired algorithms and systems is presented in [35]. The review concluded that most of the work in this area started in the very last few years and the main researched areas are continuous optimisation and travelling salesman problems.

In [40] an evaluation of the robustness of bees' foraging behaviour using a multi-agent simulation platform is presented. The study showed that the foraging strategy of a honeybee colony is robust and adaptive and that its emerging features allow the colony to find optimal solutions.

In [41] a honey-inspired algorithm is proposed to dynamically allocate Internet servers to client requests with the objective to maximise the revenue of an Internet hosting centre. The performance of the algorithm was compared with three algorithms: omniscient, greedy and static optimal. As expected, the omniscient algorithm outperforms all three algorithms, but it is significantly time and space intensive. The bee algorithm performs the best of the other two algorithms. However, the main drawback of this algorithm is that assessing the profitability of a server does not rely on local and easily calculated information in order to preserve the simplicity and efficiency of bee algorithms. Instead, a server needs to compare its total revenue rate with the overall revenue rate of the hosting centre before making any scheduling decision which is computationally expensive and time consuming.

In [42] a bee colony optimisation algorithm for a job shop scheduling problem is proposed. As indicated in Chapter 4, job shop scheduling is concerned with certain kinds of problems where each job needs to visit certain machines in a predefined order. The objective was to minimise the makespan. The algorithm goes into successive iterations to find the schedule with the highest profitability (the minimum makespan). The algorithm was compared with an ant colony and Tabu search algorithms. Results showed that Tabu search outperformed both, but the bee algorithm performed better than the ant algorithm. However, this bee algorithm is of a clairvoyant static policy, assuming that characteristics of computational jobs and resources are known in advance, which usually cannot be assumed in practice. Additionally, the algorithm is computationally expensive. In each iteration, the profitability of a schedule is compared to the average profitability of all other schedules. Neither this algorithm nor [41] have considered the tremble dance in their implementations.

5.8 Conclusion

In this chapter, a detailed analysis of the NAP is presented with algorithmic and queuing models. These models are utilised as the basis for developing HoPe with a direct correspondence at finer levels of detail to the NAP. HoPe is designed with a self-scheduling policy to conceal the resource management complexity from the personal user. It employs a decentralised cooperative and adaptive scheduling policy to cope with highly dynamic environments. The non-clairvoyant scheduling policy of HoPe aims to handle the unpredictability of incoming jobs and available resources. The entire scheduling process in HoPe depends only on local and easily calculated parameters making HoPe of high potential for mobile devices. The role altering technique of HoPe has the potential to virtualise the actual number of available resources and fluctuation in them.

Reviewing the area of insect-inspired heuristics revealed that only few work has emerged in the bee-inspired resource scheduling in particular. However, it seems that bee-inspired heuristics have not been introduced to grid computing before. Therefore, HoPe has been proposed to explore the potential of bee-inspired algorithms in mitigating the grid level resource management complexity.

5.9 References

[1] T. L. Casavant, “A taxonomy of scheduling in general-purpose distributed computing systems,” *IEEE Trans. Softw. Eng.,* vol. 14, pp. 141, 1988.

[2] A.J. Chakravarti, G. Baumgartner, and M. Lauria, “Self-organizing scheduling on the organic grid,” *Int. J. High Performance Comput. Applicat.,* vol. 20, pp. 115–130, 2006.

[3] J. Leung, L. Kelly and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* Boca Raton, FL: CRC Press, 2004.

[4] R. L. Graham, “Bounds on multiprocessing timing anomalies,” *SIAM J. Appl. Math.*, vol. 17, no. 2, pp. 416-429, 1969.

[5] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey and R.L. Graham, “Worst-case performance bounds for simple one-dimensional packing algorithms,” *SIAM J. Comput.*, vol.3, pp. 299–325, 1974.

[6] K. Li, “An average-case analysis of online non-clairvoyant scheduling of independent parallel tasks,” *J. Parallel Distrib. Comput.,* vol. 66, no. 5, pp. 617-625, May 2006.

[7] T.D. Braun and H. J. Siegel, “Heterogeneous Distributed Computing,” in *Encyclopaedia of Electrical and Electronics Engineering*, J. G. Webste, Ed. New York, NY: John Wiley & Sons, 1999.

[8] J. J. Bartholdi and P. K. Loren, “Heuristics based spacefilling curves for combinatorial problems in euclidean space,” *Manage. Science*, vol. 34, no. 3, pp. 291-305, 1988.

[9] E. Bonabeau, M. Dorigo and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. New York, NY: Oxford University Press, 1999 [Online]. Available: http://portal.acm.org/citation.cfm?id=328320.

[10] A. Abraham, C. Grosan, V. Ramos, Eds. *Stigmergic Optimization.* Springer, 2006.

[11] E. Bonabeau and C. Meyer, “Swarm intelligence: A whole new way to think about business,” *Harvard Bus. Rev.*, vol. 19 no.5, pp.107–114, May 2001.

[12] K. M. Passino, *Biomimicry for Optimization, Control, and Automation*. Springer, 2004.

[13] T. D. Seeley, *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies*. MA: Harvard University Press, 1995.

[14] A. Bader and E. Ekici, “Throughput and delay optimization in interference-limited multihop networks,” in *Proc. 7th ACM int. Symp. MobiHoc*, 2006, pp. 274-285.

[15] C. Miao, J. Weng, A. Goh, Z. Shen and B. An, “Fuzzy cognitive maps for dynamic grid service negotiation,” *Multiagent Grid Syst.,* vol. 2, pp. 101-114, 2006.

[16] A. Eryilmaz, R. Srikant, and J. Perkins, "Stable scheduling policies for fading wireless channels," *IEEE/ACM IEEE Trans. Autom. Control*, vol. 13, no. 2, pp. 411–424, Apr. 2005.

[17] R. G. Askin and C. R. Standridge, *Modeling and Analysis of Manufacturing Systems*. Wiley, 1993.

[18] V. Frisch, *The Dance Language and Orientation of Bees*, *Cambridge University Press*, 1967.

[19] F. Ruttner, *Biogeography and Taxonomy of Honeybees*. *Springer*, 1987.

[20] R. F. A. Moritz and E. E. Southwick, *Bees as Superorganisms: An Evolutionary Reality*. Springer, 1992.

[21] T. D. Seeley, S. Camazine and J. Sneyd, "Collective decision-making in honey bees: How colonies choose among nectar sources," *Behavioral Ecology Sociobiol.*, vol. 28, pp. 277-290*,* 1991.

[22] D. Sumpter and S. Pratt, "A modelling framework for understanding social insect foraging," *Behavioral Ecology Sociobiol.*, vol. 53, pp. 131-144*,* 2003.

[23] P. Navrat, "Bee hive metaphor for web search," in *Proc. CompSysTech*, Jun. 2006, Veliko Turnovo, Bulgaria.

[24] M. D. Cox and M. R. Myerscough, "A flexible model of foraging by a honey bee colony: the effects of individual behaviour on foraging success," *J. Theoretical Biol.*, vol. 223, no. 2, pp. 179-197, Jul. 2003.

[25] M. Campos, E. Bonabeau, G. Theraulaz, J. L. Deneubourg, "Dynamic Scheduling and Division of Labour in Social Insects," *Adaptive Behaviour*, vol. 8, no. 2, pp. 83-92, 2001.

[26] G. F. Oster and E. O. Wilson, Caste and Ecology in the Social Insects. Princeton: *Princeton University Press*, 1978.

[27] MAGNET Consortium, IST-MAGNET Project, Annex I, "Description of work", Nov. 2003.

[28] MAGNET Consortium, Deliverable 2.1.1, "Conceptual secure PN architecture", 2004.

[29] Condor Project [online]. Available: http://www.cs.wisc.edu/condor, [accessed Feb. 2, 2010].

[30] Legion: A Worldwide Virtual Computer [online]. Available: http://legion.virginia.edu/, [accessed Feb. 2, 2010].

[31] DSTC Nimrod/G [online]. Available: http://www.csse.monash.edu/~sgaric/nimrod/, [accessed Feb. 2, 2010].

[32] Z. Ma (Sam) and A. W. Krings, "*Insect sensory system inspired computing and communications*," *Ad Hoc Networks*, vol. 7, no. 4, pp. 742-755, 2009.

[33] M. Dorigo, *Ant Colony Optimization*, Cambridge, UK: MIT Press, 2004.

[34] R. C. Eberhart and Y. Shi, "Particle swarm optimization :developments, applications and resources,". in *Proc. IEEE Congr. Evol. Comput.* 2001,

pp.81-86.

[35] A. Baykasoglu, L. Özbakır, P. Tapkan, “Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem,” in *Swarm Intelligence, Focus on Ant and Particle Swarm Optimization*. Chan, Felix T.S. and Tiwari, Manoj Kumar, Eds. Vienna, Austria: I-Tech Education and Publishing, Dec. 2007.

[36] A. Colorni, M. Dorigo, V. Maniezzo and M. Trubian, “Ant system for job-shop scheduling,” *Belgian J. Operation Research, Stat. Comput. Science*, vol. 34, pp. 39-53, 1994,

[37] R.S. Chang, J.S. Chang and P. S. Lin, “An ant algorithm for balanced job scheduling in grids,” *Future Generation Comput. Syst.*, vol. 25, no.1, pp. 20-27, 2009.

[38] Y. Li, “A bio-inspired adaptive job scheduling mechanism on a computational grid,” *Int. J. Comput. Sci. Network Security*, vol. 6, no. 3, Mar. 2006.

[39] N.P. Lemmens, “To bee or not to bee: A comparative study in swarm intelligence,” M.S. Thesis, Maastricht University, Maastricht ICT Competence Centre, Institute for Knowledge and Agent Technology, Maastricht, Netherlands, 2006.

[40] R. Thenius, T. Schmickl and K. Crailsheim, “The ‘dance or work’ problem: why do not all honeybees dance with maximum intensity”, in *Proc. 4th CEEMAS 2005*, Budapest, Hungary, pp. 246-255.

[41] S. Nakrani and C. Tovey, “On honey bees and dynamic server allocation in internet hosting centers,” *Adaptive Behavior - Animals, Animats, Softw. Agents, Robots, Adaptive Syst.*, vol. 12, no. 3-4, pp. 223-240, 2004.

[42] C. S. Chong, A. I. Sivakumar, Malcolm, and K. L. Gay, “A bee colony optimization algorithm to job shop scheduling,” in *Proc. 38th WSC,* Dec. 2006, pp. 1954-1961.

Chapter 6
Evaluation and Results

6.1 Introduction

The aim of PM-Grids is to support a personal user with a general purpose resource-rich computing environment that is beneficial in different aspects of his/her every day and working life. This support is via sharing the user's own resources in a PAN extended with clusters of remote devices which belong to his/her PN, as detailed in Chapter 3.

Resource sharing is inevitably problematic. Therefore an efficient resource scheduler is the core of a PM-Grid. In Chapter 5 the design of a resource scheduler for PM-Grids (HoPe) is presented. In this chapter, an experimental evaluation study of HoPe is presented in a simulated PM-Grid environment at different scales.

Among the main contributions to this chapter are PM-Grid simulated models of different scales, a controlled experimental study to prove the concept of the proposed paradigm (PM-Grid) and to evaluate the performance of the proposed scheduling heuristic (HoPe), as well as performance models for the HoPe and the OSH.

In section 6.2 evaluation objectives are indicated. The detailed experimental design is described in section 6.3. In section 6.4 the resource scheduling framework proposed in Chapter 4, is applied to describe the scheduling model of HoPe and OSH. The PM-Grid simulator is presented in section 6.5. Section 6.6 explains how performance models of both heuristics were predicted. Section 6.7 describes the evaluation experiments in detail. In section 6.8 the results and performance models are illustrated and discussed. Section 6.9 concludes the chapter.

6.2 Evaluation Objectives

The thesis is that a PM-Grid can allow personal users to seamlessly combine their own personal devices, either mobile or stationary, to accomplish relatively large computational jobs. To test this thesis, an adaptive self-scheduling heuristic, HoPe, has been proposed with a non-clairvoyant scheduling policy. Such a resource scheduler is the core of PM-Grids.

HoPe, which is inspired by the nectar collection technique in honeybee colonies, is based on the hypothesis that if a colony of honeybees is able to efficiently allocate its members among nectar sources and dynamically adapt itself to environmental changes through simple non-intelligent agents, then a technology system constructed on similar principles should be able to efficiently allocate its members to job sources and automatically adapt itself in a highly dynamic environment such as PM Grids.

The end aim of the evaluation process was to evaluate the PM-Grid as a proof-of-concept. Measuring the potential and usefulness of a grid system is nothing more than evaluating its ability to efficiently schedule its underlying resources. Therefore a well-controlled experiment has been conducted on a PM-Grid model employing the purposely developed heuristic, HoPe, to schedule its resources. The aim has been fulfilled through the following objectives:

- Test HoPe performance by exploring how it is affected by variations in PM-Grid environment specifications, namely:
    - The job interarrival time: The system should sustain various loads as personal users' requirements vary significantly.
    - The number of nodes: the system should be sufficiently scalable to accommodate different infrastructure scales, as PM-Grids can be utilised by individuals as well as small size organisations, as described in Chapter 3.
- Evaluate HoPe efficiency by comparing it to a well-established heuristic in the same area, OSH, as well as an optimum value or worst bound, when possible, for each performance metric.
- Build performance models for both heuristics to obtain a clearer insight into HoPe behaviour.

6.3 Experimental Design

There are two main limitations in the simulation methodology of current scheduling research. First, there are no simulation standards and, second, traditional computing platform standards are no longer valid for modern platforms [1]. To overcome this problem, strictly controlled experiments in a logical network model of PM-Grids have been designed which involved the following steps:

1. Identifying the critical elements inherent in the design of grid scheduling systems and deciding on the set to be considered in this experiment: job interarrival time, number of nodes, job size and processor capacity.
2. Varying the experimental variables, job interarrival time and number of nodes, to simulate a representative sample of grid environments.
3. Controlling extraneous variables, job size and processor computational capacity, by randomisation to ensure a representative sample in all experiments.
4. Identifying a benchmark algorithm. The opportunistic scheduling heuristic (OSH) has been selected for this purpose.
5. Identifying suitable performance measures, stability, net throughput, mean TT and speedup, to compare HoPe and OSH.
6. Building a flexible PM-Grid simulator that offers an easily controlled environment and robust experimental design.
7. Comparing the performance of both HoPe and OSH to optimum values or worst bounds, then reporting and analysing the main findings.
8. Improving the accuracy of the simulation-base study through:
    - Running 10 simulations and accepting the mean outcome.
    - Ignoring simulation results generated in the first 60 seconds.
    - Measuring the uncertainty in data using the measure of standard deviation (SD) and displaying the values as error bars in all charts.
    - Calculating the absolute error and relative error to examine the quality of obtained results.

As addressed in [2], an approximation or heuristic algorithm A is probabilistically evaluated by comparing its solution values $A\ (I_n)$ with the optimum value $OPT\ (I_n)$ based on one of the two evaluation criteria:

- Absolute error: The absolute error is defined as the difference between the approximate and optimal solution values:

$$a_n = A\ (I_n) - OPT\ (I_n) \tag{6.1}$$

- Relative error: the relative error is defined as the ratio of the absolute error and the optimal solution value:

$$r_n = (A\ (I_n) - OPT\ (I_n))/\ OPT\ (I_n) \tag{6.2}$$

6.4 Resource Scheduling Framework in PM-Grids

This section describes the resource scheduling framework in PM-Grids, based on the framework proposed in Chapter 4, which is summarised in Table 6.1.

6.4.1 Resource model

In resource scheduling frameworks, the resource model is used to describe the nature of individual resources that can be assigned jobs. In PM-Grids this applies to executer and composer devices. As described in Chapter 3, both groups of devices belong to the same category, worker devices. Hence, they have quite similar basic capabilities such as processor capacity. However, PM-Grid devices are not dedicated grid resources; only idle cycles can be utilised in PM-Grids. This results in dynamic processor capacities over time. Therefore, the resource of PM-Grids can be viewed as parallel unrelated processors.

Heterogeneity in processor capacity is modelled assuming three types of processors (P_a, P_b, P_c) which differ only in capacities, as shown in Table 6.2. During running time, a uniform random number R_{proc} from 1 to 3 is generated to indicate the processor capacity which conforms with similar lines of research conducted by [3].

A simulation model of the PM-Grid platform is developed using Opnet^{TM} 12.0 [4] modeller. Three representative infrastructure scales of PM-Grids in potential application areas were considered:

- Small (4 workers/cluster).
- Medium (8 workers/cluster).
- Large (16 workers/cluster).

Table 6.1: Resource Scheduling Framework in PM-Grids

		Features				Scheduler	
						HoPe	OSH
Resource model		Parallel	Identical				
			Uniform				
			Unrelated			✓	✓
		Dedicated	Flow shop				
			Open shop				
			Job shop				
Job model		Independent	BoT				
			DL			✓	✓
		Dependent	DAG				
			Non-DAG				
Per. met.		Job centric				✓	✓
		Resource-centric				✓	✓
		Economy-based					
Scheduler model	Organis.	Centralised					
		Decentralised					
		Distributed	Cooperative			✓	
			Non-cooperative				✓
		Non-distributed					
	Pro.	Optimum					
		Sub-optimum	Approximation				
			Heuristic			✓	✓
	Policy	Stochastic					
		Deterministic	Non-clairvoyant			✓	✓
			Clairvoyant	Static			
				Dynamic	Batch		
					Immediate		

The model is simulated as a logical network, that consists of *N=5* clusters, as shown in Figure 6.1. All clients were placed in one cluster (cluster 0) which represents the PAN with the user at its inner core submitting jobs to his/her PM-Grid via devices in this cluster. For simplicity, the result-space is placed alone in a separate cluster (cluster 4). All other clusters consist of one work-space and *w* workers. However, as this is a logical network, device placement has no impact on the system performance.

Table 6.2: Experimental Processor Capacity

Processor	Processor capacity (Mflop/sec.)
P_a	100
P_b	50
P_c	10

Table 6.3 and Table 6.4 present the number of devices in each role as well as the number of workers in the three PM-Grid scales respectively. From the total number of workers, 75% are initially assigned an executer role, and the remaining 25% are assigned a composer role. This selection is aimed to conform with the natural distribution of roles in a honeybee colony where [5] stated that nearly 75% of honeybees are food foragers.

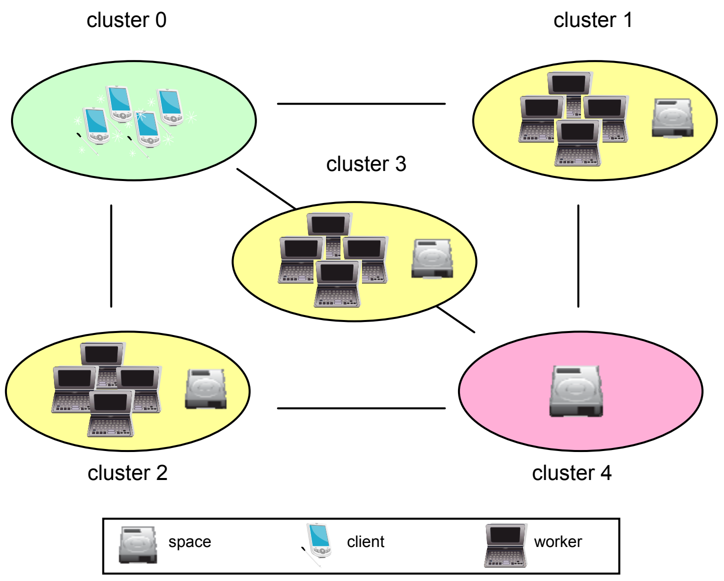


Figure 6.1: PM-Grid Model (4 workers/cluster)

This model can scale easily and allows the testing of HoPe performance in isolation of possible effects caused by physical hardware, network topology and implementation technologies. This isolation is important to gain a clear insight into HoPe performance. Experimenting with realistic networks is left for future work to see how physical hardware and network parameters of a PM-Grid may affect HoPe performance.

Table 6.3: Number of Devices in each PM-Grid Device Role

Role	No. of machines
Client	4
Work-space	3
Result-space	1
Worker	12, 24, 48

Table 6.4: Number of Workers in each PM-Grid Scale

Workers/cluster	Total no. of workers	Initial no. of executers	Initial no. of composers
4	4 × 3 = 12	8	4
8	8 × 3 = 24	18	6
16	16 × 3 = 48	36	12

6.4.2 Job model

The job model assumed by HoPe is DL applications where each job can be divided into an arbitrary number of independent tasks of low granularity, as described in Chapter 3. It is assumed that the input to each task is a single file which is sent with the task. Each task produces exactly one output file, as shown in Figure 6.2. This model can be found in many everyday application areas related to personal users such as image processing, database searching and cryptography.

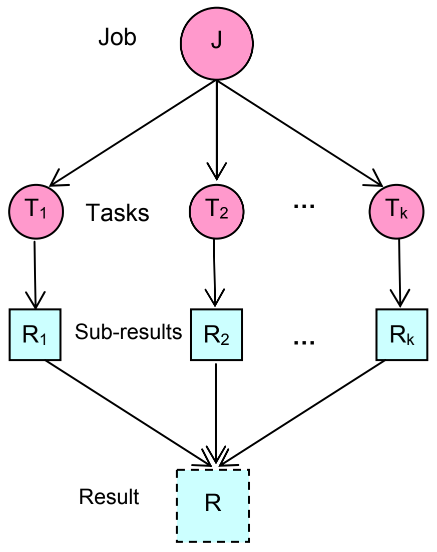


Figure 6.2: PM-Grid Job Model

Without loss of generality, this thesis has considered a cryptography application in particular as it has potential applications in personal environments where security and privacy are critical issues. The basic idea behind cryptosystems stems from the presumed difficulty of factoring large integers. The problem of factoring large integers has attracted considerable research interest. The Fundamental Theorem of Arithmetic (the Unique Prime Factorisation Theorem) stated that: every positive integer greater than one has a unique prime factorisation [6], for instance 1674=31×3×3×3×2. However, the theorem provides no insight into the factoring process itself.

During the past two decades several general purpose algorithms have been proposed to tackle this problem such as Pollard Rho Algorithm, Lenstra_s Elliptic Curve Algorithm and Trial Division Algorithm [7]. The Trial Division Algorithm is the least complex to understand and to implement [8] making it a viable integer factorisation option for devices with limited resources. Additionally, this algorithm is extremely amendable for parallelisation where each parallel processor can be assigned a number of iterations [6], making it appropriate for distributed environments. Therefore, it has been selected for experimenting with HoPe. The main steps in the simplest form of a trial division algorithm implemented in C^{++}, is shown in Figure 6.3.

```
int n;
double temp,
cin>> n;       // number to be factored
for (int i=2; i <= sqrt((double)(n)); i++)
   {
     temp=(double)n/(double)i;
     if (temp == (int)temp)
          cout<< i << ‶, ‶ ;
   }
```

Figure 6.3: Simple Trial Division Algorithm (C^{++})

The Trial Division Algorithm tries to find all positive integer divisors less than or equal to *n* (number to be factored). Clearly, it is only worthwhile to test candidate factors less than *n*. Specifically, the trial factors need go no further than $\sqrt{n}$. The algorithm execution time is a function of *n*. In the worst case, the algorithm can take up to $\sqrt{n}/2$ which gets even worse for large n. Therefore several other refinements have been suggested to enhance the algorithm. For instance, if n is odd, then only odd divisors are considered. Actually, only prime numbers need to be considered. Hence, the algorithm can be provided with a list of primes to check against it. However, all these refinements complicate the algorithm and lengthen its logic. Therefore, for most significant factoring concerns, such as public key cryptography, other factoring algorithms are more efficient.

The Trial Division Algorithm is considered as a kind of DL job model with a single iteration of the "for loop" as the smallest atomic operation. As indicated in Chapter 4, the selection of chunk size, i.e. the number of iterations constituting a task is an important issue to consider when dealing with DL models. While a small chunk size magnifies the scheduling overhead, a large chunk size imbalances the load [9]. In HoPe, the chunk size is considered as a local decision made dynamically by each worker device based on its current state.

In HoPe implementation all worker devices are assumed to have a word-size of, at least, sixteen-bits. The last prime that fits into a sixteen-bit unsigned integer should be less than 2^{16}-1=65,535 which is 65,521. That suffices to factorise numbers up to $65{,}521^2 = 4{,}293{,}001{,}441$.

Workload selection is notably arduous [10]. While real workloads and logs from real grid systems are realistic, they are designed for very specific systems and user communities, dramatically limiting their applications. On the other hand, simulated workloads, although non-realistic, are more flexible and efficient at early stages of development, and they provide the basis for cost and time wise evaluation.

Therefore, in PM-Grid evaluation, the workload model of the entire system is simulated as streams of jobs arriving at each work-space according to a Poisson process. Multiple values for both job size and interarrival time are considered to ensure a representative sample in all experiments, as necessitated by [11]. The job size is considered as an intrinsic variable and controlled by randomisation. Heterogeneity in job size is modelled assuming three sizes of jobs (J_a, J_b, J_c). During running time, a uniform random number R_{job} from 1 to 3 is generated indicating the job size, as shown in Table 6.5.

Job interarrival time is considered as an experimental variable; nine different values for interarrival time were selected in the range between two extreme cases of the expected usage of PM-Grids: (4, 8, 12, 16, 20, 32, 40, 80, 120 and 180) seconds.

Table 6.5: Experimental Job Sizes

Job	Job size (j) in Mflop
J_a	$2\times10^2 < j \le 3\times10^2$
J_b	$1\times10^2 < j \le 2\times10^2$
J_c	$j \le 1\times10^2$

6.4.3 Performance Metrics

As detailed in section 5.4.3, it is invariably difficult to achieve a compromise between scheduling performance measures. Therefore, new performance measures, that help to optimise other performance measures, are required for capturing the tradeoffs and a methodology is needed where these measures are separately observable. Hence, in this thesis the following performance measures are observed separately and their results are reported and analysed:

- Stability: where the system strives to maximise the job collection rate subject to minimising the difference between job collection and result generation rates. In this thesis stability is calculated as the absolute value of the difference between the job collection rate $F(N_c)$ and the result generation rate $F(N_p)$ as follows:

$$Stability = (1-|\, F(N_p) - F(N_c)\, |\,)\times100 \qquad (6.3)$$

- Mean turnaround time (TT): which represents the elapsed time from when a client submits a job until the client receives the corresponding results, and is calculated as:

$$TT = result\ received\ time - job\ submission\ time \qquad (6.4)$$

- Net throughput: Net throughput represents the amount of work completed by the system over a period of time. It is measured as the number of jobs completed from time zero to time *t*.

- Speedup: The speedup refers to how much a parallel system is faster than a corresponding sequential system, and is calculated as:

$$S_p = T_1 / T_p \qquad (6.5)$$

$$T_p > 0$$

where:

p is the number of processors

T_1 is the execution time of the sequential algorithm

T_p is the execution time of the parallel algorithm with p processors.

Although speedup is usually calculated based on one job, in the case of HoPe and OSH, calculating the speedup in this way would be out of context, as these heuristics operate in a steam of jobs. Therefore, the mean time of speedup is considered.

6.4.4 Scheduler Model

Two scheduling heuristics have been implemented to experiment with the PM-Grid model: HoPe and the OSH. Both are implemented using C^{++} to be compatible with the simulator platform.

Both scheduling heuristics were implanted with best effort policy, that is, no guarantee of quality of service (QoS). Therefore, once assigned, tasks do not migrate between resources and no attempts are made for rescheduling, co-scheduling, resource reservation etc. However, this issue of enhancing HoPe with some QoS strategies is left for future research. This section investigates the design features of both heuristics, HoPe and OSH, based on the framework proposed in Chapter 4.

6.4.4.1 HoPe

As detailed in Chapter 5, Hope is a specifically tailored heuristic to meet the scheduling requirements of PM-Grids and identical environments where resources are dynamic and heterogeneous. HoPe has a decentralised cooperative organisation and a local, non-clairvoyant, self-management, best effort and adaptive scheduling policy.

Basically, HoPe can be considered as a kind of guided search heuristic, hence we expect it to have a performance level in between a random algorithm and an omniscient algorithm that guarantees an optimal solution if it does exist. In other words, while HoPe performance is expected to be better than a randomisation algorithm, its performance is not expected to compete with an omniscient algorithm. However, implementing an omniscient algorithm is radically time and space intensive [12]. Therefore, this thesis limits itself to contrasting HoPe with a randomisation algorithm, OSH, then it examines how far results of both algorithms vary from an optimal theoretical value or a worst bound.

6.4.4.2 Opportunistic Scheduling Heuristic (OSH)

The OSH is a general purpose resource scheduling heuristic that assigns each job in an arbitrary order to the next available machine without considering the execution time of the task on the machine. OSH takes a greedy assignment strategy in which no processor is idle if there are more jobs to run. The main reason for selecting OSH is the negligible amount of knowledge about the running environment and jobs which makes it suitable to the PM-Grid environments and therefore comparable with HoPe. Additionally, the OSH is the most used heuristic in high throughput computing resource management systems such as Condor [13]. It is argued that in many cases, simple scheduling approaches such as greedy algorithms are viable alternatives and are preferable in practice to more sophisticated algorithms as they are as effective, more robust, more scalable and simpler to implement [14].

There are as many implementations of OSH as there are numbers of systems using it. While all have the basic idea explained above, they vary in implementation details. For instance, in [15] a centralised scheduler examines all machines to find the machine that becomes ready next. In [16] idle machines assign jobs to themselves by accessing a shared queue of jobs. In [14] a DAG-based implementation for the OSH is presented.

The main drawback of the OSH is its poor load balancing performance. However, since, the considered job model is the DL, any OSH implementation for such a job model should consider the chunk size selection problem. Hence, in this thesis the OSH has been implemented with a variable chunk size self-scheduling scheme, where the chunk size each processor assigns to itself dynamically changes based on the processor capacity. The adjustment of chunk size implies better load balancing performance [9].

6.5 PM-Grid Simulator

Initially, a real test-bed was constructed to evaluate PM-Grid design as detailed in [17]. However, it was found that experimenting with such a real test-bed in early evaluation stages is extremely difficult due to the need to test the proposed design in isolation from implementation technologies as well as the need to continuously alter and tune hardware and software parameters. Therefore, a simulation based approach has been followed which is widely used for evaluating and studying grid scheduling systems due to its configurability and repeatability [18].

There are some simulation packages that have recently emerged to simulate grid environments, such as GridSim [19] and Simgrid [20]. However, these simulation packages are designed for traditional grid environments. PM-Grid platform has special characteristics and requirements that are totally different from traditional grids. Furthermore, deploying HoPe instead of the default resource management systems of these simulation packages requires altering their core modules which is extremely difficult and time consuming. Therefore, all experiments in this thesis are carried out on a purposely developed PM-Grid simulator. This approach of utilising a purposely built grid simulator to meet specific research goals comes in line with [21, 22].

PM-Grid models were built using the network simulator Opnet^{TM} 12.0 [4]. Opnet is a commercially available modelling and simulation tool to simulate computer networks using the finite-state modelling concept, as shown in Figure 6.4 . It comes with a number of built-in models for nodes, routers, servers among others. Using these models, one can easily simulate many kinds of networks and analyze their performance. However, using Opnet to simulate grid systems is not straight forward. All functions related to having the networked nodes functioning as a grid system and cooperate together in solving computational problems needed to be coded manually in C and C++ and incorporated with Opnet network models.

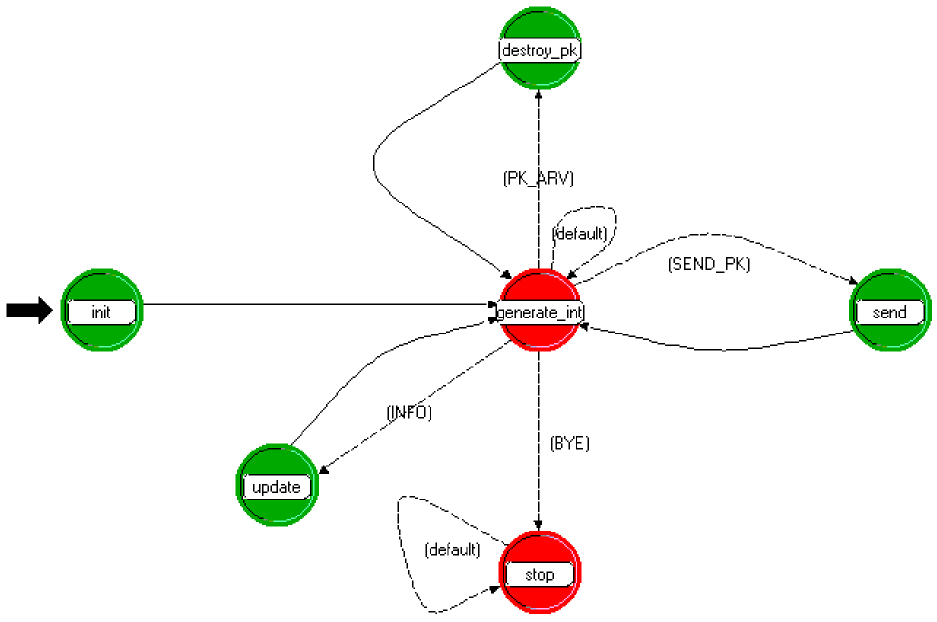


Figure 6.4: Finite-State Modelling Concept - Opnet

The PM-Grid simulator is built out of four main modules that run on a logical PM-Grid model, as illustrated in Figure 6.5. Each module can be easily altered to maximise the simulator flexibility and alleviates future research with it. The four modules are:

1. Job generator module: This module randomly generates jobs based on the job model detailed in section 6.4.2. The output of this module is fed to the workload generator module.
2. Workload generator module: This module generates the entire workload to be processed by the PM-Grid simulator. The workload is modelled as dynamic continuous streams of jobs submitted from client devices to work-spaces based on the workload model described in section 6.4.2.
3. Processor capacity modeller module: This is a simple module that randomly generates an integer number to indicate the processor capacity of a worker device based on the processor capacity model described in section 6.4.1.
4. Resource scheduler module: This module contains the logic of the two heuristics, HoPe and OSH. Based on the running experiment, one of the two heuristics is selected to allocate jobs to workers.

The PM-Grid model is simulated at three different scales, as described in section 6.4.1. Selected performance metrics are fed to the PM-Grid model to generate performance data that is used to evaluate the efficiency of both heuristics.

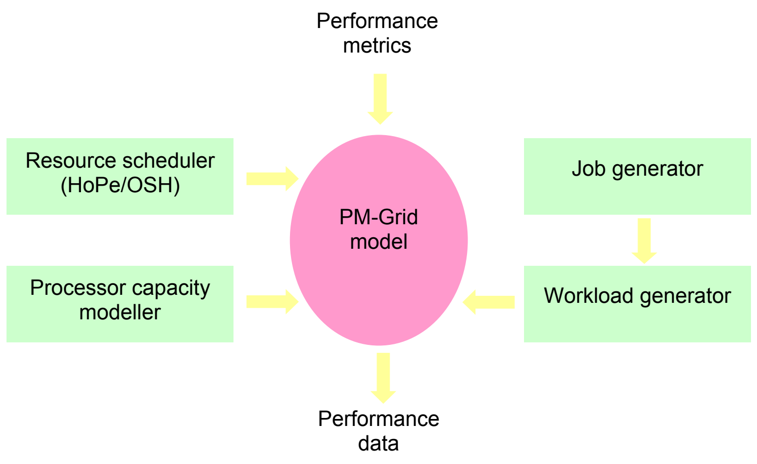


Figure 6.5: PM-Grid Simulator

6.6 Performance Models

Mathematical and graphical performance models that predict HoPe and OSH behaviours, under different running conditions, are generated using multiple regressions. EREGRESS [23, 24], Microsoft Excel Add-In software was used to:

- Predict mathematical performance models using multiple regressions and full quadric equations. The models which include linear, quadratic, and cross terms have the following general form:

$$\begin{aligned} \textit{Performance-measure} = b_0 + b_1 \times \textit{interarrival\_time} + b_2 \times \textit{grid\_scale} \\ + b_3 \times \textit{interarrival\_time} \times \textit{interarrival\_time} \\ + b_4 \times \textit{interarrival\_time} \times \textit{grid\_scale} \\ + b_5 \times \textit{grid\_scale} \times \textit{grid\_scale} \end{aligned} \qquad (6.6)$$

- Generate a 3D graphical model for each predicted mathematical performance measure model.
- Analyse the results using the ANOVA test.

The statistical significance of the full quadratic models predicted was evaluated using Fisher's statistical test (*F*) and F-significant *(F-signif.)*. Large *F* values and low values of *F-signif* indicate a high model significance. An *F-signif* value of 0.05 indicates a significant model at the 95% significance level. The significance and the magnitude of the estimated coefficients of each variable and all their possible linear and quadratic interactions on the performance of both HoPe and OSH were determined. Coefficients with effects less than 95% of significance (*P-value* less than 0.05) play a critical role in the performance measure model equation.

The results of the significance tests on the model and its coefficients are listed in tables, such as Table 6.9. The first row shows the predicted mathematical model. The second raw presents the values of *F* and *F-signif.* The first column indicates the coefficient that is computed and the second column shows its value. The third column is the *P-value*. The fourth and fifth columns show the –95% and +95% confidence values respectively for a particular response coefficient.

6.7 Experiments

To evaluate HoPe, two main issues were considered:

- Scalability to a larger number of nodes.
- Sustainability under different loads.

Two context parameters were controlled to simulate representative samples of the PM-Grid environment:

- Number of workers per cluster (grid scale): Three PM-Grid infrastructure scales were considered: small (4 workers/cluster), medium (8 workers/cluster) and large (16 workers/cluster).
- Job interarrival time: The interarrival time represents the time difference between successive arrivals of jobs. Values of interarrival time were selected in the range of two extreme cases of the expected usage of PM-Grids: (4, 8, 12, 20, 32, 40, 80, 120 and 180) sec.

Hence the total number of created scenarios is 2 × 9 × 3 = 54 scenarios (number of heuristics × number of values for interarrival times × number of values for workers per cluster). Data related to four performance metrics, stability, mean TT net throughput and speedup, were measured for each scenario.

Extraneous variables, job size and processor computational capacity, were randomised to ensure representative samples in all experiments. Heterogeneity in processor capacity was modelled assuming three types of machines (P_a, P_b, P_c) with different capacities. Heterogeneity in job size was modelled assuming three types of jobs (J_a, J_b, J_c) with different sizes. During running time, a uniform random number R_{proc} from one to three is generated describing the processor capacity and another random number R_{job} following the same distribution is generated to describe job size heterogeneity. The processor capacity and job size were generated based on similar lines of research conducted by [23]

Jobs were generated by four clients with a Poisson process and exponential interarrival times with means (4, 8, 12, 20, 32, 40, 80, 120 and 180) sec. Computational jobs were implemented as DL applications to factor large integers (up to 4,293,001,441). Each job is contained in one packet and produces one output file. For simplicity, the communication cost to send a packet from one machine to another is not considered at this stage. It is assumed that one machine can process only one operation at a given moment (resource constraints) and once task started, operation runs to completion (no pre-emption condition).

The selection of values for empirical parameters of HoPe presented in section 5.6.2 are presented in Table 6.6.

Table 6.6: Values of HoPe Empirical Parameters

Parameter	Usage	Value
C	Executer help threshold (*EHT*)	4
D	Composer help threshold (*CHT*)	1
E	Compose help message duration(*CHMD*)	1

6.8 Results, Performance Models and Discussion

Each scenario, simulating five hours (18000 sec.) of real time, ran ten times and means were calculated after discarding data from the initial 60 sec. Results are displayed as bar charts for ease the comparison. The uncertainty in values using the standard deviations, are displayed as a vertical line superimposed on each bar.

6.8.1 Stability

Stability results, and discussion of these results as well as performance models, are presented in the following sections.

Table 6.7: Total Arrival Rates

Mean interarrival time (sec.) 1 source	Total arrival rate (job/sec.) 4 sources (λ`)
4	1.000
8	0.500
12	0.333
20	0.200
32	0.125
40	0.100
80	0.050
120	0.033
180	0.022

6.8.1.1 Results

Figures 6.6 and Figures 6.7 illustrate HoPe and OSH stability respectively, in terms of the difference in rate between job collection and result generation cycles calculated using the mean time. Each figure consists of three sub-figures which demonstrate the behaviour of the corresponding heuristic, at the three grid scales, 4, 8 and 16 workers /cluster, when compared to an optimal value. This value is calculated as:

$$\lambda` = \sum_{i=1}^{4} \lambda_I \qquad (6.7)$$

where:

$\lambda`$ is the total job arrival rate at the four sources that corresponds to each interarrival time, as illustrated in Table 6.7.

Table 6.8 illustrates the absolute and relative errors in stability measure under HoPe and OSH.

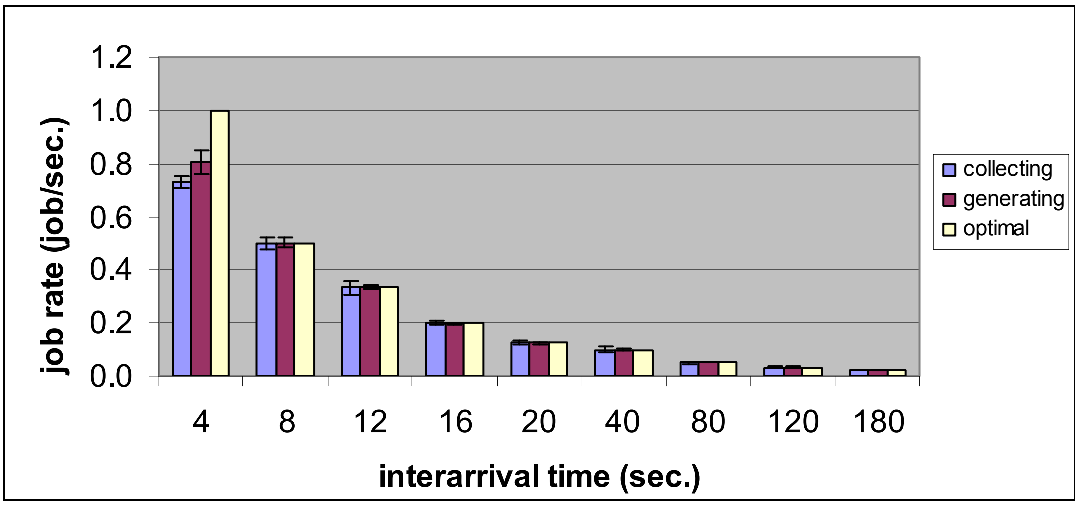
6.8.1.2 Discussion

Figure 6.6 shows that in general, HoPe is able to maintain both job collection and result generation cycles in balance at optimal rates, indicating a stable system in more than 95% of the experimental scenarios. However, under extremely heavy loads (interarrival time = 4 sec. and a grid scale = 4 workers/cluster) the system is less stable. This phenomenon relates to the impact of extremely heavy loads in increasing both the volume of remaining work in work-spaces and the waiting time experienced by each executer. Large values of remaining work and waiting time stimulate devices to repeatedly altering their roles which negatively reflects on stability. However, even in this situation, HoPe shows better stability performance than the OSH as illustrated in Figure 6.7.

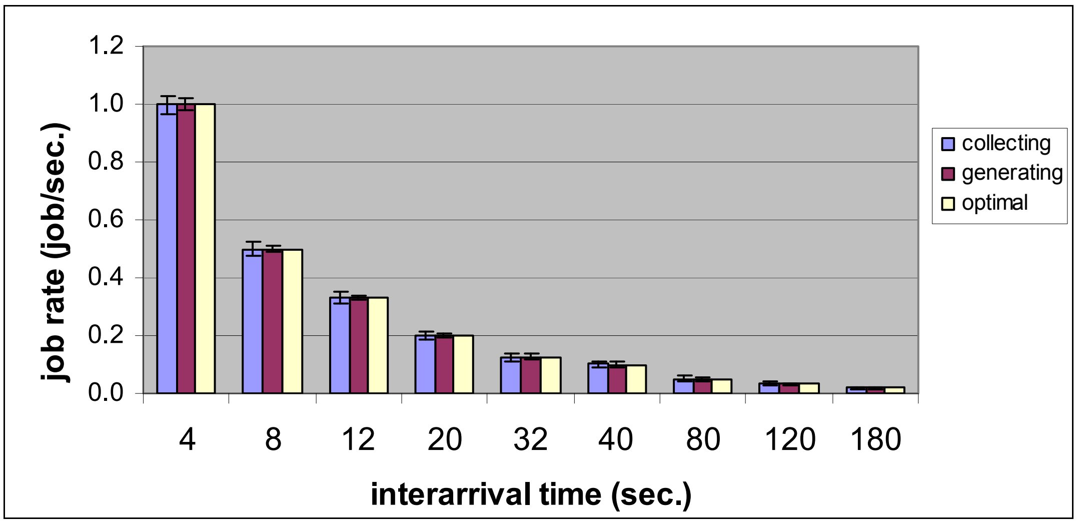
An important observation is that increasing the number of workers per cluster from 4 to 8 tends to enhance the stability and also increases the rates of both job collection and result generation cycles. However, this improvement discontinues when the number of workers per cluster is increased from 8 to 16. In this situation, additional workers have no visible effect in enhancing HoPe stability performance.

Table 6.8: Stability Absolute and Relative Errors

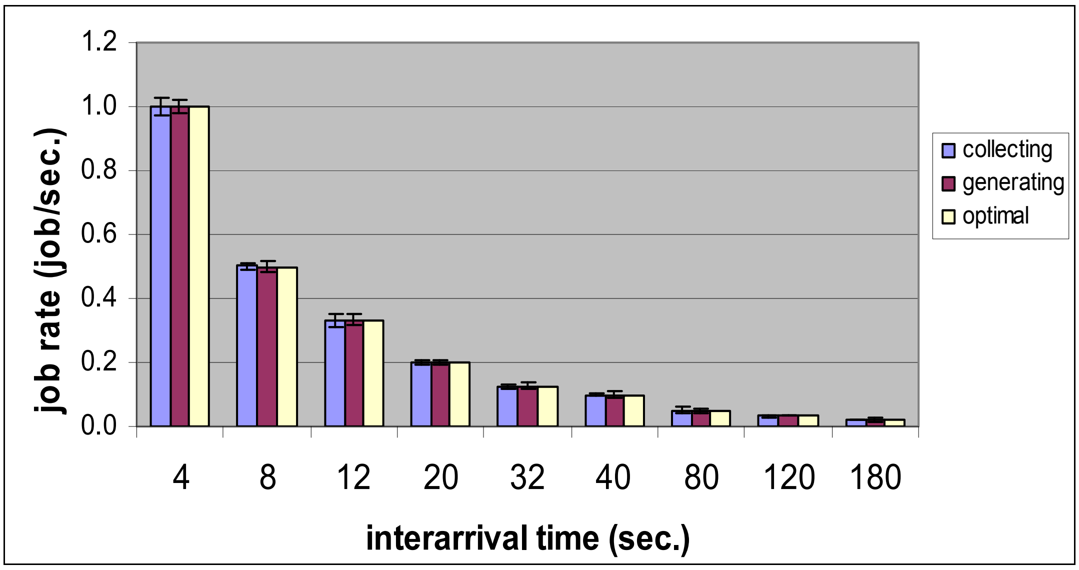
Grid scale	Interarrival time	Absolute error		Relative error	
		HoPe	OSH	HoPe	OSH
4 workers/cluster	4	-7.5	-5.4	-0.075	-0.054
	8	-0.1	-4.1	-0.001	-0.041
	12	-0.1	-2.5	-0.001	-0.025
	20	0	-1.2	0	-0.012
	32	0	0	0	0
	40	0	0	0	0
	80	-0.1	0	-0.001	0
	120	0	0	0	0
	180	0	0	0	0
8 workers/cluster	4	-0.1	-8.6	-0.001	-0.086
	8	-0.1	-2.5	-0.001	-0.025
	12	-0.1	-0.4	-0.001	-0.004
	20	0	-0.3	0	-0.003
	32	0	-0.2	0	-0.002
	40	0	-0.1	0	-0.001
	80	-0.2	0	-0.002	0
	120	-0.1	0	-0.001	0
	180	0	0	0	0
16 workers/cluster	4	-0.1	-5.5	-0.001	-0.055
	8	-0.1	0	-0.001	0
	12	-0.3	0	-0.003	0
	20	0	0	0	0
	32	0	0	0	0
	40	-0.1	0	-0.001	0
	80	0	0	0	0
	120	0	0	0	0
	180	0	0	0	0



(a) 4 workers per cluster

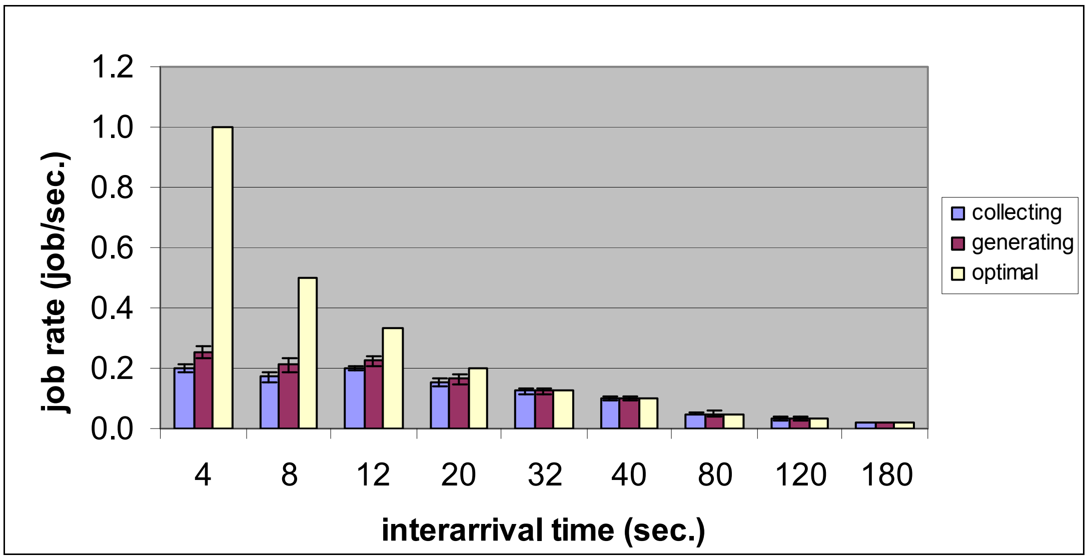


(b) 8 workers per cluster

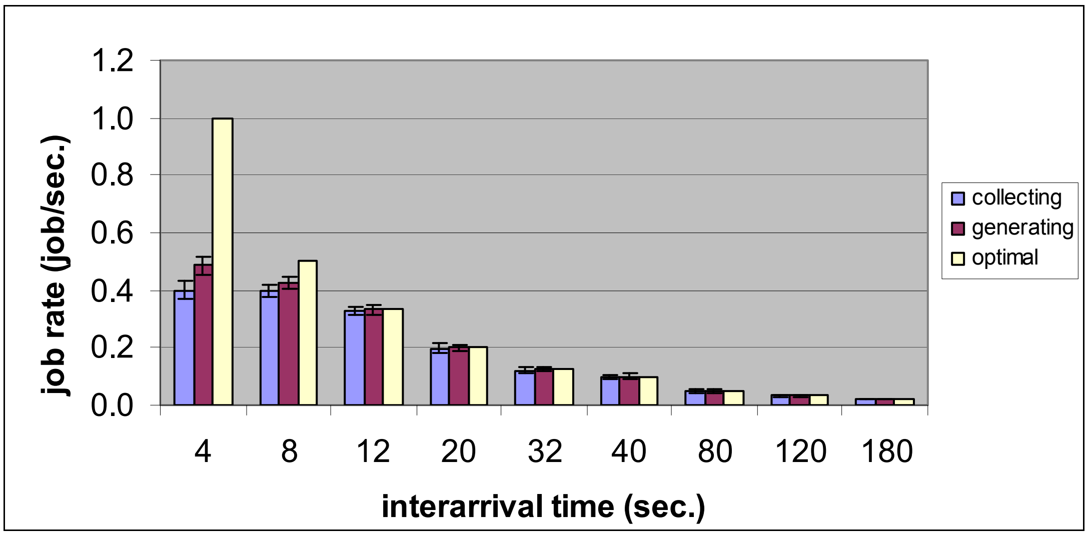


(c) 16 workers per cluster

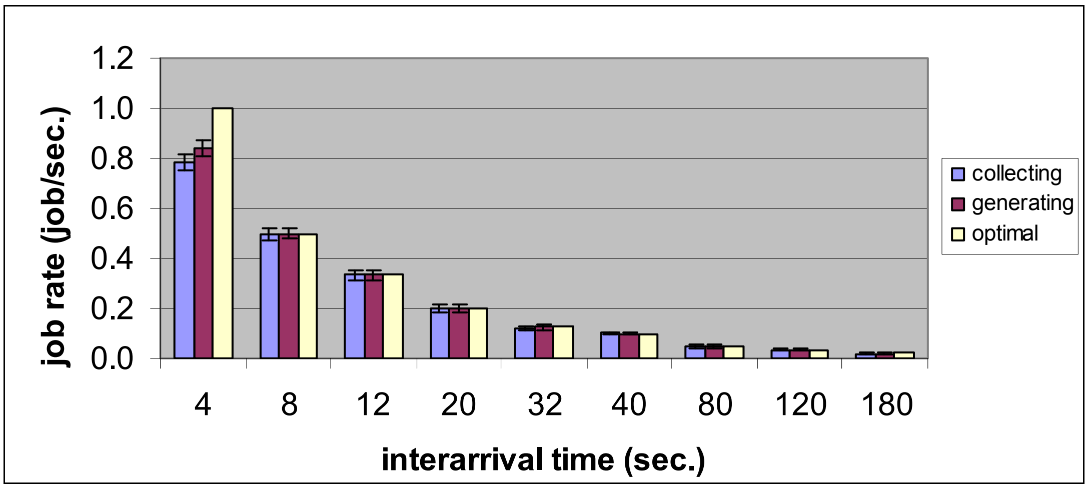
Figure 6.6: HoPe Stability



(a) 4 workers per cluster



(b) 8 workers per cluster



(c) 16 workers per cluster

Figure 6.7: OSH Stability

This is because optimum rates have been already achieved with 8 workers/cluster. For instance when interarrival time = 4 sec., the total job arrival rate from four clients is:

$$\lambda` = \sum_{i=1}^{4} \lambda_i = 4\times 1/4 = 1 \text{ job /sec.}$$

When interarrival time = 8 sec.

$$\lambda` = \sum_{i=1}^{4} \lambda_i = 4\times 1/8 = 0.5 \text{ job/sec}$$

Hence, to maintain optimum stability, the job collection and result generating rates should not exceed 1 and 0.5 for interarrival times 4 and 8 sec., respectively, regardless of the number of workers.

Figure 6.7, which illustrates the stability performance of OSH, shows that rates of job collection and result generation are not in balance particularly for short interarrival times ($4 \leq$ interarrival times ≤ 20 where grid scale = 4 workers/cluster*;* $4\leq$ interarrival times ≤ 8 where grid scale = 8 workers/cluster; and interarrival times = 4 where grid scale = 16 workers/cluster). However, for longer interarrival times, OSH shows balance, indicating a stable system in nearly only 75% of the experimental scenarios. In contrast to HoPe, the OSH rates of job collection and result generation continue to increase for short interarrival times, as the grid increases in scale since optimal values are still to be reached.

6.8.1.3 Stability Models

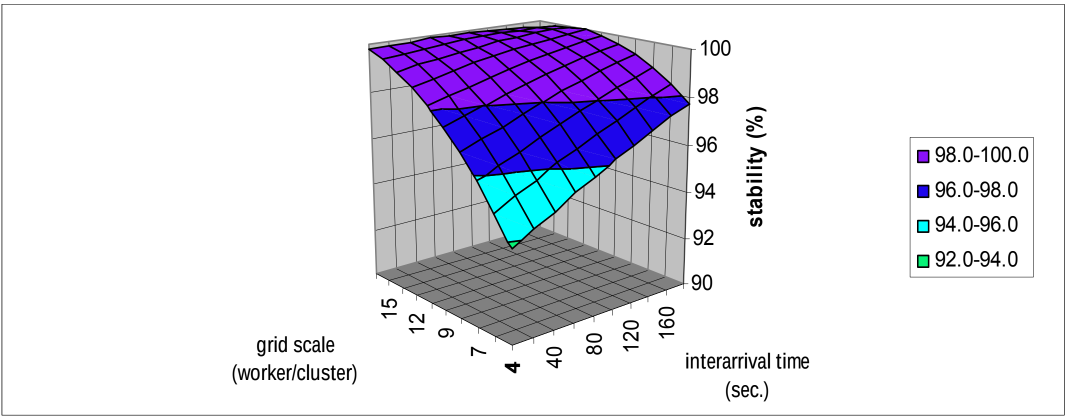
Stability performance models of HoPe and the OSH are shown in the 3D sub-figures (a) and (b) respectively of Figure 6.8. The models show the general stability behaviour of both heuristics when interarrival time falls in the range from 4 to 180 sec. and the grid scale is in the range from 4 to 16 workers/cluster. The stability is calculated using the absolute value of the difference between job collection and result generation rates as provided in equation 6.7:

$$Stability = (1 - |job\ processing\ rate - result\ generating\ rate|) \times 100 \qquad (6.7)$$

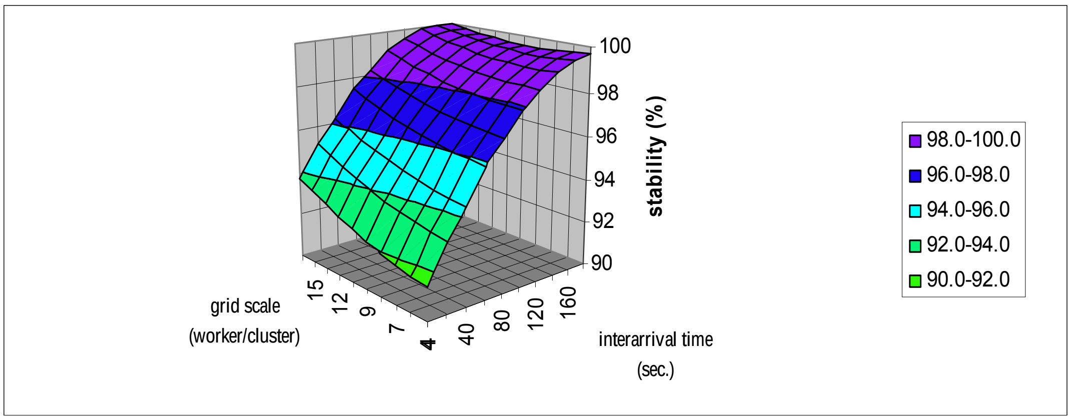
The model in the sub-figure (a) shows that HoPe tends to maintain optimum stability (100%-98%) in a considerably wide area of the entire problem space. As expected, when there are enough workers, no matter how often jobs arrive, HoPe can maintain the difference between job collection and result generation at a minimum level. The situation changes gradually as the grid scale shrinks when stability becomes more sensitive to the interarrival time. The insignificance *P-value* of all coefficients, in Table 6.9, emphasises that HoPe has successfully marginalised the effects of variations in the grid scale and the job interarrival time when stability is considered.

The model in sub-figure (b) shows that the OSH tends to maintain optimum stability in a relatively small area of the entire problem space. It is also clear from the model, and also from the significance *P-value* of (b_1 and b_3) coefficients in Table 6.10, that the OSH is more sensitive to variations in the interarrival time under all grid scales in the displayed range.

Mathematical equations and statistical data of the HoPe stability model and the OSH stability model are presented in Table 6.9 and Table 6.10 respectively.



(a) HoPe



(b) OSH

Figure 6.8: Stability Models

Table 6.9: Statistical Data of HoPe Stability Model

HoPe_stability = $b_0 + b_1 \times$ *interarrival_time*+ $b_2 \times$ *grid_scale* + b_3 *interarrival_time* × *interarrival_time* + $b_4 \times$ *interarrival_time* × *grid_scale* + $b_5 \times$ *grid_scale* × *grid_scale*				
F =1.018 *F-signif* = 0.376				
Coefficients		***P-value***	**-95%**	**95%**
b_0	89.80	3.04E-24	86.16	93.43
b_1	0.03430	0.155	-0.00885	0.07744
b_2	1.085	0.268	0.262	1.909
b_3	-0.000022	0.339	-0.000236	0.000194
b_4	-0.001900	0.378	-0.00392	0.00016
b_5	-0.02900	0.385	-0.06805	0.01009

Table 6.10: Statistical Data of OSH Stability Model

OSH_stability = $b_0 + b_1 \times$ *interarrival_time*+ $b_2 \times$ *grid_scale* + b_3 *interarrival_time* × *interarrival_time* + $b_4 \times$ *interarrival_time* × *grid_scale* + $b_5 \times$ *grid_scale* × *grid_scale*				
F = 3.159 *F-signif* = 0.02789				
Coefficients		***P-value***	**-95%**	**95%**
b_0	90.80	7.94E-22	86.09	95.51
b_1	0.09511	0.00385	0.03921	0.151
b_2	0.03720	0.960	-1.025	1.100
b_3	-0.000239	0.01638	-0.000517	3.86058E-05
b_4	-0.000977	0.432	-0.00362	0.00167
b_5	0.00800	0.836	-0.04264	0.05864

6.8.2 Throughput

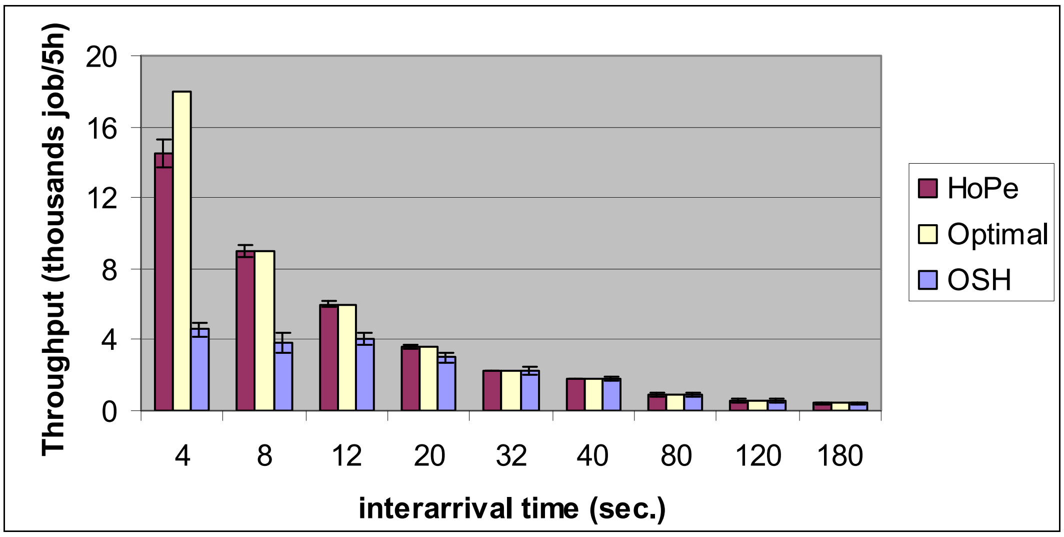
Net throughput results and discussion of these results, as well as performance models, are presented in the following sections.

6.8.2.1 Results

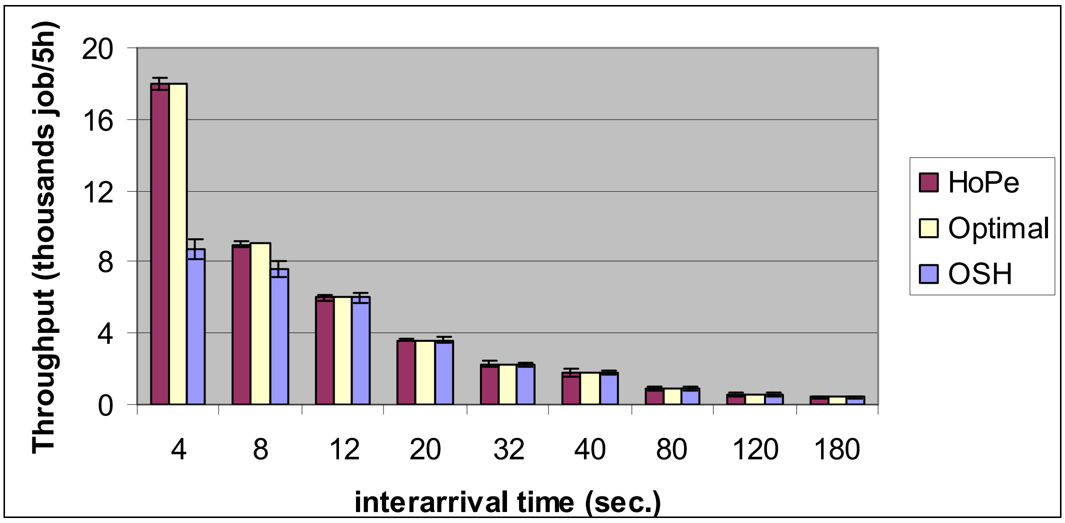
Figure 6.9 consists of three sub-figures showing both HoPe and OSH net throughput in terms of the mean number of completed jobs per five hours at the three grid scales, 4, 8 and 16 workers/cluster, compared to optimal values.

Optimal values, shown in Table 6.11, are obtained assuming that all jobs submitted to the system are completed successfully before the end of the simulation, that is, within five hours.

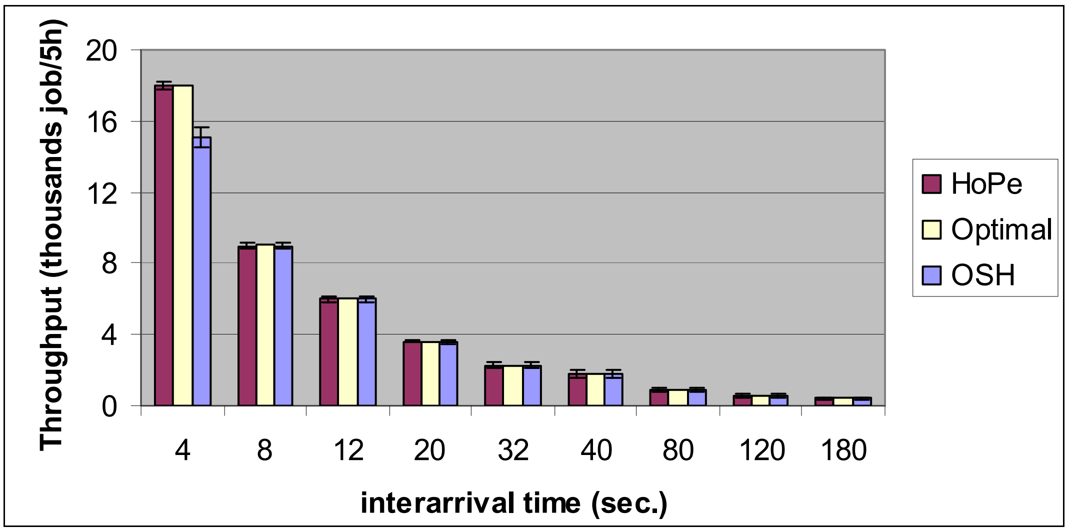
Statistical data of absolute and relative errors in net throughput measures under HoPe and OSH is illustrated in Table 6.12.



(a) 4 workers per cluster



(b) 8 workers per cluster



16 workers per cluster

Figure 6.9: Net Throughput in HoPe and OSH

Table 6.11: Optimal Net Throughput

Interarrival time 1 source (sec.)	Optimal throughput 4 sources (job/5 h)
4	18000
8	9000
12	6000
20	3600
32	2250
40	1800
80	900
120	600
180	400

Table 6.12: Net Throughput Absolute and Relative Errors

Grid scale	Interarrival time	Absolute error		Relative error	
		HoPe	OSH	HoPe	OSH
4 workers/cluster	4	-3504	-13434	-0.19467	-0.74633
	8	0	-5188	0	-0.57644
	12	-4	-1968	-0.00067	-0.328
	20	-7	-613	-0.00194	-0.17028
	32	-4	-6	-0.00178	-0.00267
	40	-3	-4	-0.00167	-0.00222
	80	-3	-4	-0.00333	-0.00444
	120	-1	-4	-0.00167	-0.00667
	180	0	0	0	0
8 workers/cluster	4	-20	-9270	-0.00111	-0.515
	8	-9	-1361	-0.001	-0.15122
	12	-8	-12	-0.00133	-0.002
	20	-4	-7	-0.00111	-0.00194
	32	-5	-5	-0.00222	-0.00222
	40	-4	-5	-0.00222	-0.00278
	80	-4	-4	-0.00444	-0.00444
	120	-4	-4	-0.00667	-0.00667
	180	0	0	0	0
16 workers/cluster	4	-16	-2900	-0.00089	-0.16111
	8	-8	-11	-0.00089	-0.00122
	12	-5	-8	-0.00083	-0.00133
	20	-5	-4	-0.00139	-0.00111
	32	-2	-5	-0.00089	-0.00222
	40	-4	-4	-0.00222	-0.00222
	80	-4	-4	-0.00444	-0.00444
	120	-4	-4	-0.00667	-0.00667
	180	0	0	0	0

6.8.2.2 Discussion

The sub-figures of Figure 6.9 show that HoPe has successfully obtained throughput equal to the optimum value in more than 95% of experimental scenarios. However, under extremely heavy load (interarrival time = 4 and grid scale = 4 workers/cluster), HoPe shows a throughput value which is less than the optimum.

Nevertheless, the throughput value of HoPe in this situation is nearly triple the value of the OSH.

The same observation reported in HoPe stability regarding the discontinued improvement in performance, when the grid scale is increased from 8 workers/cluster to 16 workers/cluster, is also apparent in net throughput. As the optimal net throughput has been already achieved with 8 workers/cluster, as shown in the sub-figure (b), additional workers have no visible effect in enhancing HoPe performance, as illustrated in the sub-figure (c).

On the other hand, the OSH obtained optimal throughput values only in less than 75% of experimental scenarios, under mild to light loads. OSH struggles to obtain optimal values for throughput under extremely heavy and heavy loads (interarrival time = 4, 8, 12 and 20 for grid scale = 4 workers/cluster, interarrival time = 4 and 8 for grid scale = 8 workers/cluster and interarrival time = 4 for grid scale = 16 workers/cluster). In contrast to HoPe, the OSH net throughput continues to increase as the grid increases in scale since optimal values are still to be reached.

6.8.2.3 Throughput Models

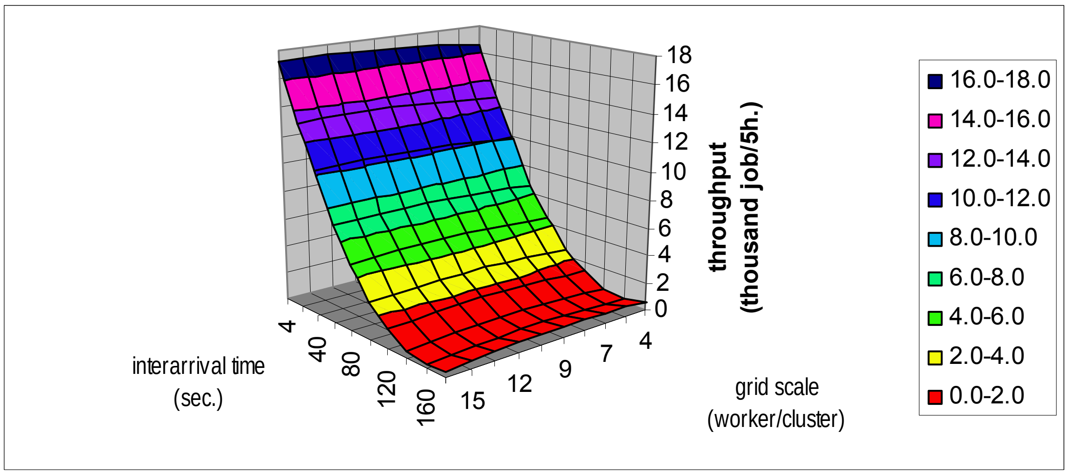
In Figure 6.10, the 3D models in sub-figures (a) and (b) summarise the behaviour of HoPe and the OSH respectively in terms of the net throughput for interarrival times in the range from 4 to 180 sec. and grid scales in the range from 4 to 16 workers/cluster. As expected, the net throughput under both heuristics tends to increase as the load inside the system becomes heavier as the interarrival time gets smaller in value.

Comparing the two sub-figures demonstrates the superiority of HoPe performance when net throughput is considered. An important observation is clear also where the net throughput of HoPe looks marginally affected by the grid scale. Consequently, the HoPe net throughput is mainly a function of the interarrival time, which clearly demonstrates the efficiency of the dynamic role-altering technique adopted by HoPe, where the system virtualises the actual number of workers to cope with the current context requirements. In contrast, the OSH net throughput is significantly affected by the grid scale, particularly for low values of the interarrival time.

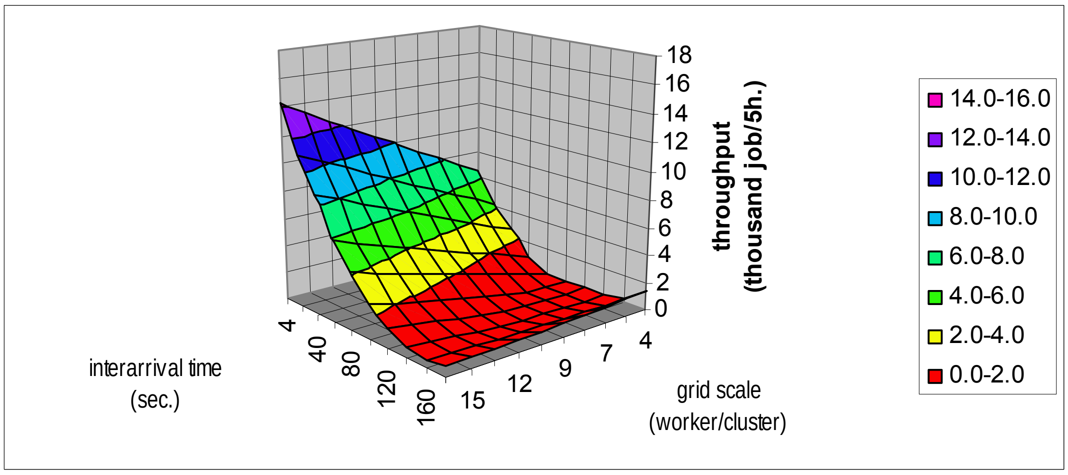
It is important not to misinterpret the models in Figure 6.10 and also the charts in

Figure 6.9; the low throughput values do not indicate poor system performance, but rather they show raw data that needs to be interpreted in context. For instance, the minimum throughput which has been achieved by both heuristics at grid scale 16 workers/cluster and 180 sec. interarrival time is in fact the maximum throughput that can be achieved in this context. Therefore, it might be more realistic, in later stages of analysis, to depict relative values of the throughput which can be calculated as a percentage of the optimal value, if known, rather than depicting raw values.

Mathematical equations and statistical data of the HoPe net throughput model and the OSH net throughput model are presented in Table 6.13 and Table 6.14 respectively.



(a) HoPe



(b) OSH

Figure 6.10: Net Throughput Models

Table 6.13: Statistical Data of HoPe Throughput Model

HoPe_throughput = $b_0 + b_1 \times$ *interarrival_time* + $b_2 \times$ *grid_scale* + b_3 *interarrival_time* × *interarrival_time* + $b_4 \times$ *interarrival_time* × *grid_scale* + $b_5 \times$ *grid_scale* × *grid_scale*				
F = 7.332 *F-signif* = 0.000409				
Coefficients		***P-value***	**-95%**	**95%**
b_0	16666.6	0.03262	7880.13	25453.2
b_1	216.63	0.822	-1766.0	2199.3
b_2	-214.58	0.000334	-318.88	-110.28
b_3	-8.016	0.862	-102.51	86.47
b_4	-0.437	0.855	-5.370	4.495
b_5	0.685	0.00120	0.167	1.204

Table 6.14: Statistical Data of OSH Throughput Model

OSH_throughput = $b_0 + b_1 \times$ *interarrival_time* + $b_2 \times$ *grid_scale* + b_3 *interarrival_time* × *interarrival_time* + $b_4 \times$ *interarrival_time* × *grid_scale* + $b_5 \times$ *grid_scale* × *grid_scale*				
F = 10.44 *F-signif* = 3.87375E-05				
Coefficients		***P-value***	**-95%**	**95%**
b_0	4540.0	0.183	-639.418	9719.334
b_1	615.25	0.288	-553.455	783.954
b_2	-116.90	0.00101	-178.383	-55.416
b_3	2.33	0.569	-53.368	58.02841
b_4	-4.000	0.09841	-6.90689	-1.0913
b_5	0.565	0.00109	0.259235	0.870749

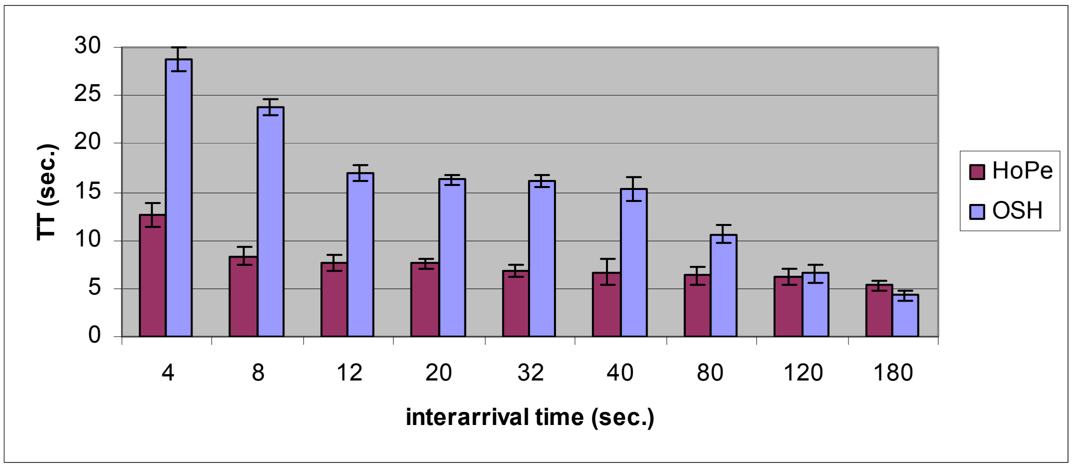
6.8.3 Turnaround Time (TT)

The experimental results of TT and the discussion of these results, as well as TT performance models are presented in the following sections.

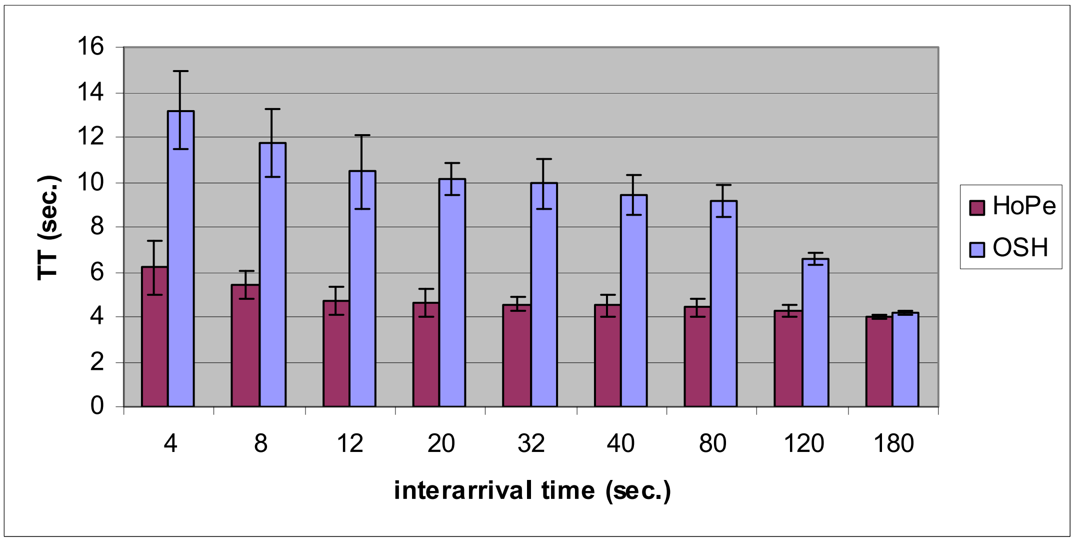
6.8.3.1 Results

Figure 6.11 consists of three sub-figures that show the TT of both HoPe and OSH, calculated based on equation (6.4), at the three grid scales, 4, 8 and 16 workers/cluster. An empirical worst bound was employed to compare with TT values achieved by HoPe and OSH.

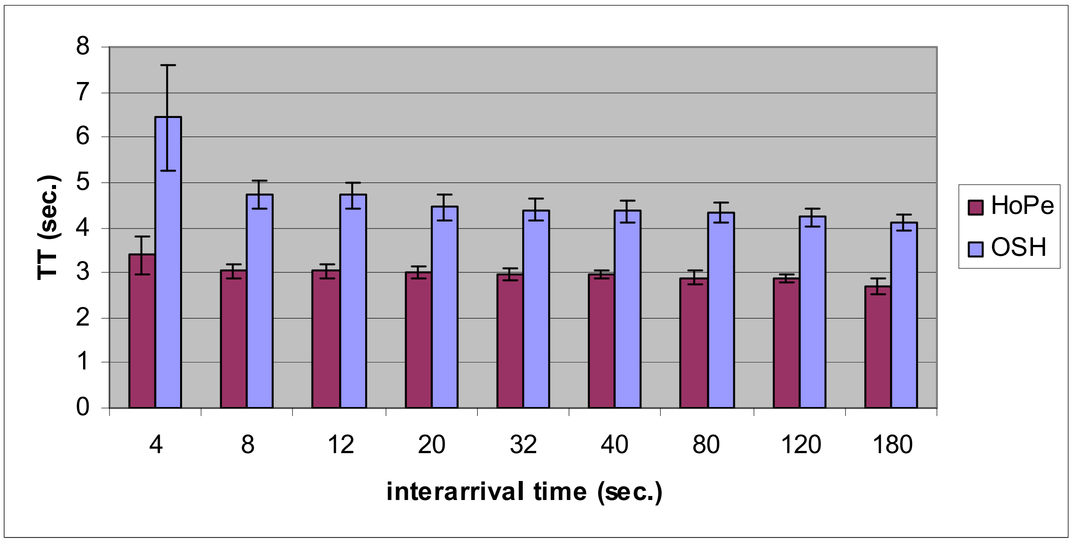
The empirical worst bound, which equals 27 sec., represents the maximum TT when a large job J_a is executed sequentially in a capacity-limited machine P_c. Although, the worst bound was calculated based on one job, in the case of HoPe and OSH, the TT was calculated using the time average per job which is the TT experienced by all jobs in the scenario divided by the number of jobs, which include the queuing delay in work-spaces and the result-space. Therefore, in practice, the mean TT can go beyond the worst bound.



(a) 4 workers per cluster



(b) 8 workers per cluster



(c) 16 workers per cluster

Figure 6.11: Mean TT in HoPe and OSH

6.8.3.2 Discussion

The three sub-figures of Figure 6.11 show the superiority of HoPe performance when TT is considered. HoPe has considerably a shorter TT than the OSH with HoPe achieving less than a half of the TT achieved by the OSH in more than 60% of all experimental scenarios. However, under very light loads (interarrival time = 120 and 180 sec. for grid scale = 4 workers/cluster and interarrival time = 180 sec. for grid scale = 8 workers/cluster) both OSH and HoPe reached nearly the same TT due to the small number of jobs in the system.

HoPe has TT values in the range from 3 to 13 which are all considerably better than the empirical worst bound. Hence, in its worst case, HoPe has a TT value which is less than the half of the maximum TT and in the best score it is nearly one tenth the maximum TT. On the other hand, the OSH has its TT values in the range from 4 to 29 which is actually beyond the worst bound in the worst case and is nearly one seventh the worst bound for the best case.

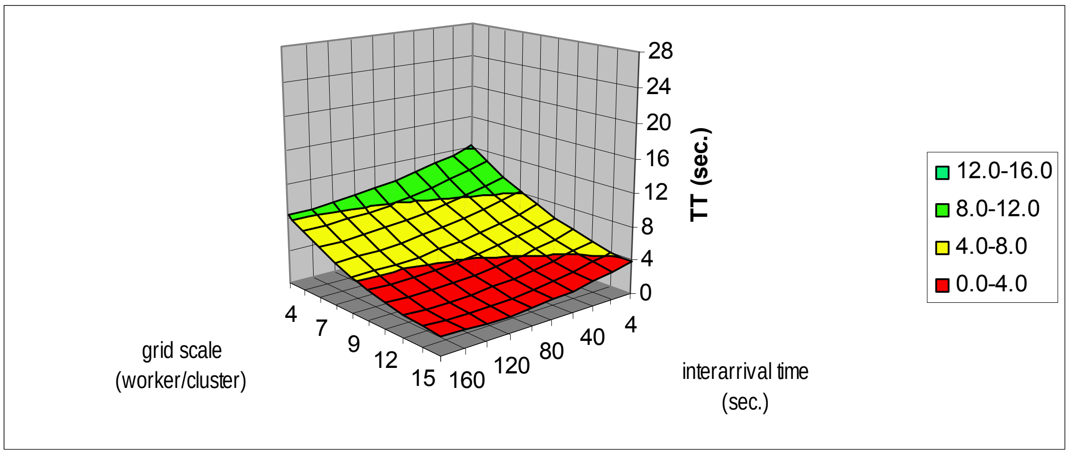
An important observation regarding the TT is that it seems to be less affected by variations in the interarrival time within the same grid scale in the case of HoPe than in the case of the OSH, which again shows the effectiveness of the dynamic role assignment technique in HoPe.

6.8.3.3 TT Models

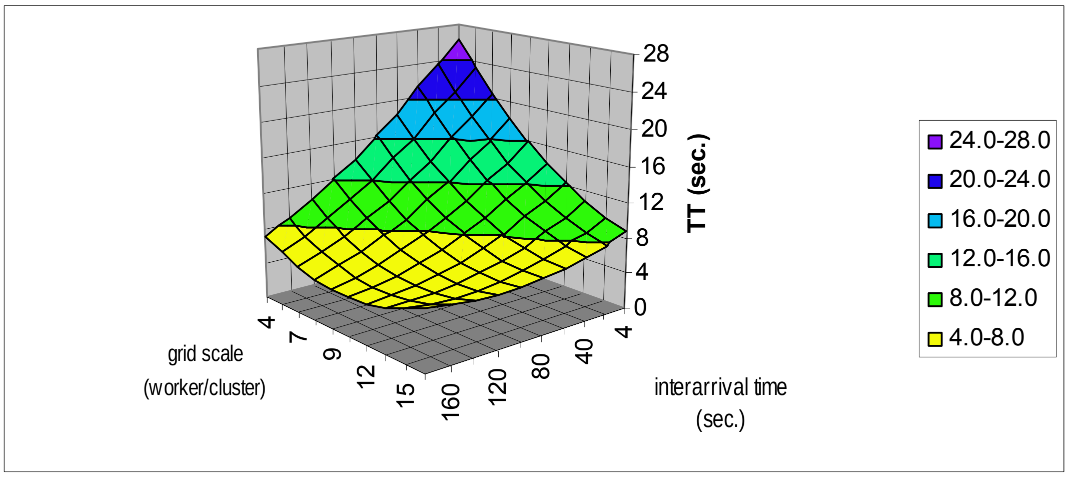
Figure 6.12 consists of two 3D sub-figures summarising the behaviour of HoPe and the OSH respectively in terms of the TT for interarrival times in the range from 4 to 180 sec. and grid scales in the range from 4 to 16 workers/cluster.

The dominance of HoPe performance is clear by comparing the scales in the TT axis in the two sub-figures. Sub-figure (a) shows that the TT value under HoPe is gradually getting smaller as the grid becomes larger while the interarrival time has notably less effect in large grid scales. The case is different when it comes to the OSH, as illustrated in the sub-figure (b), where the interarrival time has an increased effect on the value of the TT.

As expected, under both heuristics the TT approaches its minimal values as both the grid scale and the interarrival time approach their maximum values, while the TT approaches its maximum as both approach their minimum. Mathematical equations and statistical data of the HoPe mean TT model and the OSH mean TT model are presented in Table 6.15 and Table 6.16 respectively.



(a) HoPe



(b) OSH

Figure 6.12: TT Models

Table 6.15: Statistical Data of HoPe TT Model

HoPe_ TT *= b_0 + b_1× interarrival_time+ b_2 × grid_scale + b_3 interarrival_time × interarrival_time + b_4× interarrival_time × grid_scale + b_5× grid_scale × grid_scale*				
F = 22.15 *F-signif* = 1.01124E-07				
Coefficients		***P-value***	**-95%**	**95%**
b_0	17.8	3.86E-10	15.26	20.34
b_1	-1.5	0.000172	-2.074	-0.926
b_2	-0.04678	0.00408	-0.07696	-0.01660
b_3	0.03958	0.00666	0.01224	0.06692
b_4	0.0011	0.03270	-0.000328	0.00253
b_5	0.00011	0.109	-3.94256E-05	0.000261

Table 6.16: Statistical Data of OSH TT Model

$OSH\_TT = b_0 + b_1 \times interarrival\_time + b_2 \times grid\_scale + b_3\, interarrival\_time \times interarrival\_time + b_4 \times interarrival\_time \times grid\_scale + b_5 \times grid\_scale \times grid\_scale$				
F = 32.67 F-signif = 3.19009E-09				
Coefficients		***P-value***	**-95%**	**95%**
b_0	37.80	3.34E-10	31.77	43.83
b_1	-3.000	0.000280	-4.361	-1.639
b_2	-0.185	2.57E-05	-0.256	-0.113
b_3	0.07492	0.02565	0.01005	0.140
b_4	0.00820	5.48E-05	0.00482	0.01159
b_5	0.000230	0.08247	-0.00012	0.000588

6.8.4 Speedup

Speedup results and discussion of these results, as well as performance models, are presented in the following sections.

6.8.4.1 Results

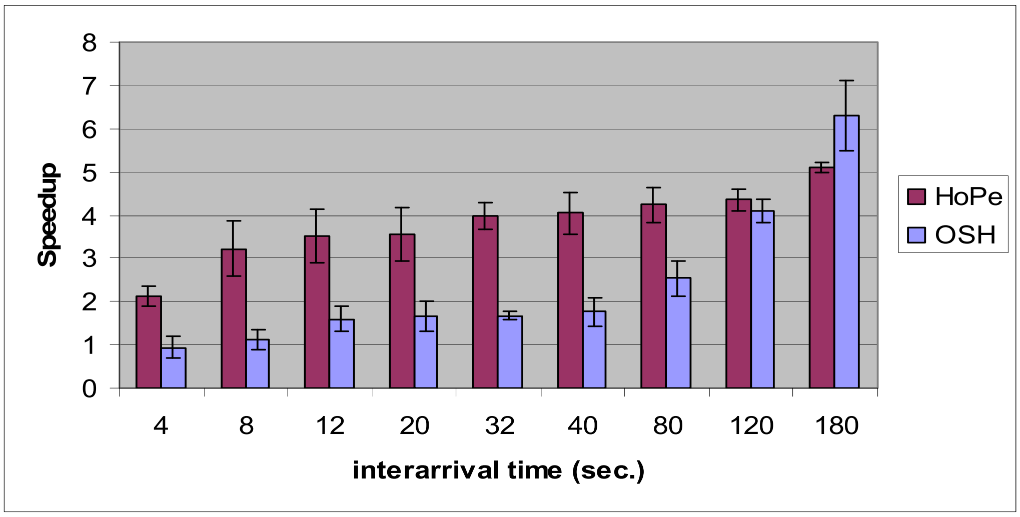
Figure 6.13 consists of three sub-figures showing the speedup of both HoPe and OSH at the three grid scales. The speedup is calculated based on equation (6.5) with the empirical value of 27 sec. as the execution time of the sequential algorithm. A worst bound of one is assumed, representing the case when both running times of executing a job sequentially, in one machine, and in parallel machines, are equal.

6.8.4.2 Discussion

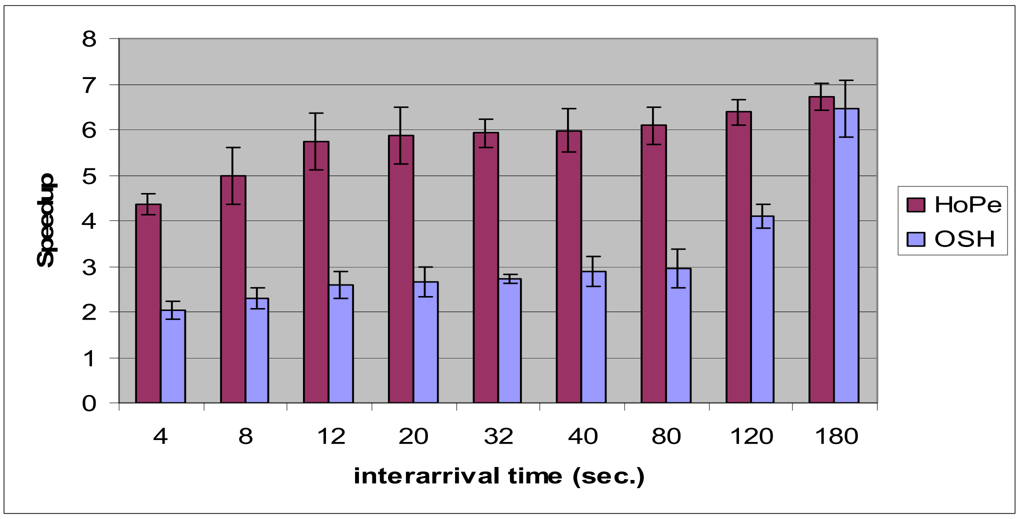
Figure 6.13 shows that HoPe has maintained a noticeably higher speedup which reaches the double speedup of the OSH in nearly 60% of all scenarios. However, the difference between the two heuristics in performance decreases gradually as the interarrival time gets larger in small and medium grid scales.

HoPe has its speedup values in the range from 2 to 10 which is double the speed of the sequential execution (worst bound) in its worst case and ten times faster than the sequential execution at best. The speedup of the OSH lies in the range from 0.9 to 7 which is a slowdown in its worst case and, in its best case, it is only seven times faster than the sequential execution.

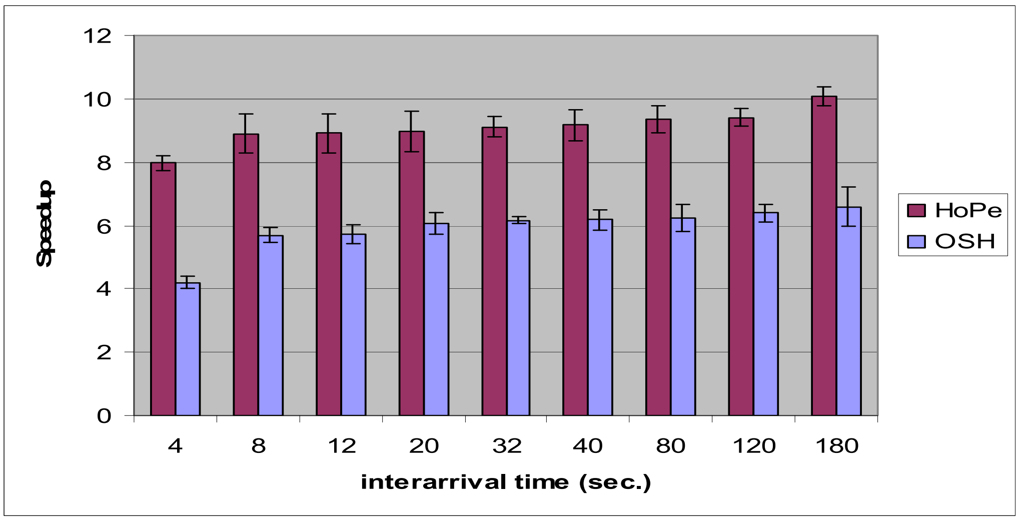
As expected, the speedup of both HoPe and the OSH is highly affected by the grid scale in terms of the total number of worker devices in the system. The interarrival time has a lower effect when HoPe is considered.



(a) 4 workers per cluster



(b) 8 workers per cluster



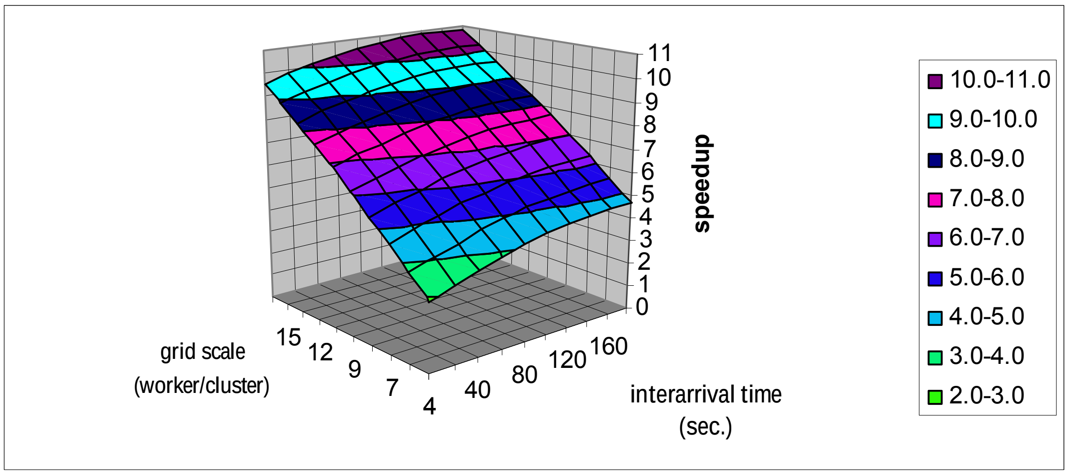
(c)16 workers per cluster

Figure 6.13: Average Speedup in HoPe and OSH

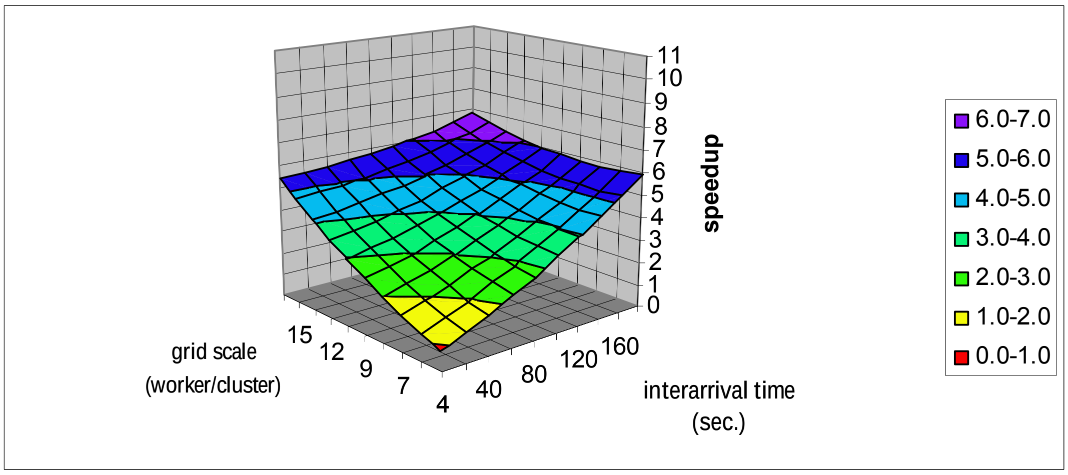
6.8.4.3 Speedup Models

The speedup of HoPe and OSH is illustrated in the three dimensional sub-figures (a) and (b) of Figure 6.14.

The dominance of the HoPe speedup over the speedup of OSH is clear from the figure which illustrates that HoPe has a considerably higher speedup in a wide area of the speedup surface. The marginalised effect of the interarrival time under HoPe is clear in the sub-figure (a) where the speedup surface has a gentle slope in the interarrival time direction in contrast to the steep slope in the sub-figure (b) in the case of the OSH. Mathematical equations and statistical data of the HoPe speedup model and the OSH speedup model are presented in Table 6.17 and Table 6.18 respectively.



(a) HoPe



(b) OSH

Figure 6.14: Speedup Models

Table 6.17: Statistical Data of HoPe Speedup Model

HoPe_ speedup = $b_0 + b_1 \times$ *interarrival_time*+ $b_2 \times$ *grid_scale* + b_3 *interarrival_time* × *interarrival_time* + $b_4 \times$ *interarrival_time* × *grid_scale* + $b_5 \times$ *grid_scale* × *grid_scale*				
F = 168.83 *F-signif* = 3.32935E-16				
Coefficients		***P-value***	**-95%**	**95%**
b_0	0.115	0.158	-0.902	1.132
b_1	0.02203	0.00106	0.00996	0.03410
b_2	0.698	2.24E-05	0.469	0.928
b_3	-5.7E-05	0.06190	-0.000117	3.1E-06
b_4	-0.000281	0.318	-0.000852	0.000290
b_5	-0.00703	0.196	-0.01796	0.00391

Table 6.18: Statistical Data of OSH Speedup Model

OSH_ speedup = $b_0 + b_1 \times$ *interarrival_time*+ $b_2 \times$ *grid_scale* + b_3 *interarrival_time* × *interarrival_time* + $b_4 \times$ *interarrival_time* × *grid_scale* + $b_5 \times$ *grid_scale* × *grid_scale*				
F = 98.84 *F-signif* = 7.31342E-14				
Coefficients		***P-value***	**-95%**	**95%**
b_0	0.003	0.693	-1.124	1.130
b_1	0.02827	0.000253	0.01489	0.04164
b_2	0.143	0.256	-0.111	0.397
b_3	4E-05	0.225	-2.7E-05	0.000107
b_4	-0.00174	1.13E-05	-0.00237	-0.00111
b_5	0.01225	0.04776	0.000133	0.02437

6.9 Conclusion

This chapter has presented the details of a controlled empirical study carried out to experiment with the PM-Grid models and evaluate HoPe performance in terms of stability, net throughput, mean TT and speedup. It has also presented the predicted performance models of both heuristics, HoPe and OSH under different running conditions of grid scale and job interarrival times.

Experimental results indicate the dominance of HoPe performance and the efficiency of its role altering technique. These results also demonstrate the ability of HoPe to considerably reduce the effect of variations in grid scale and job interarrival times, illustrating better scalability and sustainability, when compared to the OSH.

HoPe has successfully maintained optimal stability and throughput in more than 95% of the experiments with HoPe achieving three times better than the OSH under extremely heavy loads. In terms of TT and speedup, HoPe has also shown dominant performance which is twice better than the OSH performance in more than 60% of all experiments.

These promising results suggest deploying PM-Grids in real life scenarios and evaluating HoPe in other HTC systems to get better insight into their performance.

6.10 References

[1] A. Legrand, M. Quinson, H. Casanova and K. Fujiwara, "The SIMGRID project simulation and deployment of distributed applications," in *Proc. 15th IEEE Int. Symp. High Performance Distrib. Comput.,* 2006, pp. 385-386.

[2] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Eds., *Handbook on Scheduling: From Theories to Applications.* New York: Springer, 2007.

[3] CoreGRID, "Comparative evaluation of the robustness of DAG scheduling heuristics," FP6-004265, Tech. Rep. TR-0120, Dec. 5, 2007

[4] OPNET Technologies [online]. Available: http://www.opnet.com/, [accessed Feb. 2, 2010].

[5] T. D. Seeley, The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies. MA: Harvard University Press, 1995.

[6] A. Baker, *A Concise Introduction to the Theory of Numbers*, Cambridge, UK: Cambridge University Press, 1984.

[7] R.P. Brent, "Parallel algorithms for integer factorization", London Mathematical Society Lecture Note Series, vol. 154, *Number Theory and Cryptography*, J.H. Loxton, Ed., pp. 26-37, Cambridge, UK: Cambridge University Press, 1990.

[8] C. Barnes, "Integer factorization algorithms," Tech. Rep., Department of Physics, Oregon State University, Dec. 2004.

[9] W. Shih, C. Yang and S. Tseng, "A performance-based parallel loop scheduling on grid environments," *J. Supercomput.,* vol. 41, pp. 247-267, 2007.

[10] A. B. Downey and D. G. Feitelson, "The elusive goal of workload characterization," *Performance Evaluation Rev.,* vol. 26, pp. 14-29, 1999.

[11] E. Frachtenberg and D. G. Feitelson, "Pitfalls in parallel job scheduling evaluation," in *Proc. 11th Workshop Job Scheduling Strategies for Parallel Process.*, 2005, New-York, pp. 257-282.

[12] S. Nakrani and C. Tovey, "On honey bees and dynamic server allocation in internet hosting centers," *Adaptive Behavior - Animals, Animats, Softw. Agents, Robots, Adaptive Syst.*, vol. 12, no. 3-4, pp. 223-240, 2004.

[13] Condor Project [online]. Available: http://www.cs.wisc.edu/condor, [accessed Feb. 2, 2010].

[14] R. Huang, H. Casanova and A. A. Chien, "Using virtual grids to simplify application scheduling," in *Proc. 20th IPDPS 2006*, pp. 25-29.

[15] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proc. Heterogeneous Comput.*

Workshop, 1999, pp. 30-44.

[16] P. Tang, P. C. Yew, and C. Zhu, “Impact of self-scheduling on performance of multiprocessor systems,” in *Proc. 3rd Int. Conf. Supercomput.*, 1988, pp. 593–603.

[17] H. Kurdi, H. S. Al-Raweshidy, I. Khan and M. Li, “PM-Grid: a Personal Mobile Grid for ubiquitous computing environments,” Presented at *SiC07,* May 2007, Newcastle, UK.

[18] C. Jiang, C. Wang, X. Liu and Y. Zhao, “A survey of job scheduling in grids,” in *Proc. APWeb 2007/ WAIM 2007*, Huang Shan, China, pp. 419-427.

[19] GridSim [online]. Available: http://www.buyya.com/gridsim/, [accessed Feb. 2, 2010].

[20] Simgrid [online]. Available: http://simgrid.gforge.inria.fr/, [accessed Feb. 2, 2010].

[21] L. He, S. Jarvis, D. Spooner, D. Bacigalupo, G. Tan and G. Nudd, “Mapping DAG-based applications to multiclusters with background workload,” in *Proc. CCGrid*, 2005, vol. 2, pp.855-862.

[22] S. Song, K. Hwang, Y.-K. Kwok, “Trusted grid computing with security binding and trust integration,” *J. Grid Comput.*, vol. 3, pp. 53-73, 2005.

[23] D.D. Stephan, J. Werner, R.P. Yeater, Essential regression and experimental design for chemists and engineers, MS Excel Add-in Software Package, 1998–2001.

[24] J. Bulacov, J. Jirkovsky, M. Muller and R.B. Heimann, *Surf. Coat. Technol.,* vol. 201, 2006.

Chapter 7
Conclusion and Future Research

7.1 Summary

This thesis has argued that resources of personal devices, whether mobile or stationary, can be productively leveraged to service their users. By doing so, personal users will be able to ubiquitously run relatively complex computational jobs, which cannot be accommodated in their individual personal devices or while they are on the move. This has the potential of realising the ambitious grid visions of scaling grid systems to a larger number of entities and smaller devices. To this end the thesis proposes PM-Grids that superimpose grid functionality over networked personal devices.

The work in this thesis started by surveying the area of grid computing and distributed systems for paradigms relevant to PM-Grids. The survey revealed two main findings. First, there are few research projects which have addressed the mobility issue in grid computing but only at the organisational level. Second, fewer research projects have targeted grid systems at the personal level, but the focus has only been on facilitating file sharing applications. Therefore, architectural designs of PM-Grids were developed to address both personalisation and mobility issues in grid computing.

The most important aspect of realising a grid system is a scheduler that efficiently utilises its resources. However, the extremely dynamic nature, diversity and limited capabilities of resources, as well as difficulties in predicting the nature and timing of incoming jobs, are all factors that increase the complexity of the scheduling problem in PM-Grids.

Therefore, a survey on resource scheduling frameworks was conducted to address design features required for a resource scheduler that can cope with the extraordinarily difficult scheduling conditions in PM-Grids. The survey revealed that decentralised, cooperative, local, adoptive, non-clairvoyant and self-scheduling schemes are among the top requirements to deal with the complexity of this problem. Consequently, a resource scheduler, HoPe, was proposed and implemented based on these requirements. HoPe was augmented with techniques analogous to those utilised by the honeybee colony, while allocating worker bees to nectar sources under the extraordinarily difficult conditions of weather unpredictability and food variability.

Next, PM-Grid designs and HoPe implementation were evaluated thoroughly through a strictly controlled empirical study with a well-established heuristic in HTC, the OSH, as a benchmark algorithm. Comparisons with optimal values and worst bounds were conducted to gain a clear insight into HoPe behaviour under different running conditions of the number of jobs and grid scales.

Experimental results showed that HoPe has successfully maintained optimal stability and throughput in more than 95% of the experiments, with HoPe achieving three times better than the OSH under extremely heavy loads. In terms of the turnaround time and speedup, HoPe has effectively achieved less than 50% of the turnaround time incurred by the OSH while doubling its speedup in more than 60% of the experiments.

7.2 Conclusion

The overall aim of the thesis has been to introduce PM-Grids as a novel paradigm in grid computing for endowing individuals with resource-rich infrastructures that can serve as virtual general-purpose and mobile supercomputers. PM-Grids have the potential to bridge the gap between personal users and mobile devices on the one side, and current grid systems on the other.

The thesis has also aimed to address the non-clairvoyant scheduling problem in grid computing, where job information is not available to the system before the end of the execution. HoPe which is a novel honeybee inspired resource scheduling heuristic with a decentralised self-management and adaptive scheduling policy has been proposed to achieve this aim.

The thesis aims have been fulfilled resulting in the following seven main contributions:

First, architectural designs and models for PM-Grids have been developed based on the PNs architecture and as a natural extension to them; an abstract layered view, a detailed inside view and simulated models have been presented and evaluated at different scales in terms of the numbers of jobs and devices per cluster.

Second, a detailed design, implementation and evaluation of HoPe have been initiated. To the best of our knowledge, HoPe is the first algorithm to shed light on the non-clairvoyant scheduling problem in grid computing. It is the first honeybee-inspired algorithm attempting to solve the resource scheduling problem relying totally on local and computationally simple parameters.

Third, a queuing theory with a simulation based approach to the NAP modelling from the resource scheduling perspective has been initiated. A generic model for the NAP has been created as a queuing network, which is simulated in several representative scenarios. Furthermore, detailed algorithmic analysis and modelling of the NAP have been presented with honeybee techniques that had not been considered in previous work.

Fourth, a comprehensive taxonomy of grid systems has been proposed. Such a comprehensive taxonomy, which has not been presented in previous work, is significant for studying grid systems under one framework and assisting detailed comparisons between them. It also aids in understanding current research trends in grid computing and anticipating future trends attempting to establish a solid background in the rapidly evolving area of grid computing.

Fifth, a framework for resource schedulers has been proposed with a unified presentation of previously published taxonomies. Such a framework is deemed necessary to amalgamate the area of resources scheduling under a common, uniform set of nomenclatures and terminologies.

Sixth, a controlled empirical evaluation framework to prove the concept of PM-Grids and to evaluate the performance of HoPe has been developed. A flexible simulator has been built for this purpose allowing the control of experimental parameters, randomising extraneous variables as well as measuring and analysing various performance metrics.

Seventh, performance models of HoPe and OSH have been predicted in forms of mathematical equations and 3D graphical representations. These models are important to gain a clearer insight into the behaviour of each heuristic in regard to stability, net throughput, turnaround time and speedup under various running conditions of job interarrival times and grid scales.

It can be concluded, based on the experimental results and predicted performance models, that using HoPe for resource scheduling in PM-Grids considerably reduced the effect of variations in grid scale and job interarrival times, illustrating better scalability and sustainability, when compared to the OSH.

These results recommend considering the deployment of PM-Grids in real life scenarios and the utilisation of HoPe in other parallel processing and high throughput computing systems. Much work remains to be done but the potential benefits are considerable. It is hoped that this thesis contributes in some measure to realising the futuristic grid visions.

7.3 Future Research

After experimenting with PM-Grid models and evaluating HoPe performance, it can be confidently said that the results are encouraging. However, these accomplishments need to be followed with thorough development efforts to transform the PM-Grid models into reality and apply HoPe in other contexts beyond PM-Grids. The work in this thesis opens up research on various interesting issues and directions.

7.3.1 Short Term Future Research

In the short term future research, the following issues need to be explored.

7.3.1.1 PM-Grids

It is important to note that this thesis has not emphasised implementation details as the aim at this stage was to demonstrate a “proof-of-concept” of PM-Grids.

In the long term, there might be a need for interaction between PM-Grids and other grid systems. Designing PM-Grids with this possibility in mind facilitates future interaction. Therefore, careful selection of implementation technology for a middleware system for PM-Grids is important at an early stage of future work.

For instance, the latest release of Jini technology [7] from Sun Microsystems [8] allows applications to be easily packaged as services that are available across a shared Java space. Both Jini and Java spaces have the potential of realising PM-Grids and assisting the interaction with other Grid middleware systems.

7.3.1.2 HoPe

One of the strengths of HoPe lies in the adaptive role altering technique that it has successfully implemented, where worker devices automatically exchange their roles during the running time based on the current system context. However, currently, initial device roles are manually assigned to devices at the initialisation phase. Automatic role assignment, based on device features, would need to be considered to further augment the self-management property of HoPe.

Additionally, it is anticipated that users would specify time limits or priorities for their jobs to which the scheduler should adhere. Constraint- and priority-based scheduling are important features to be added to enhance the design of HoPe.

7.3.1.3 Stability Performance Measure

This thesis has maintained the implicit assumption that stability can help to optimise both the TT and throughput and can capture the tradeoffs between them more efficiently than a multi-objective function that gives each performance measure a certain weight.

The evaluation results have demonstrated that HoPe, which uses stability as the only scheduling objective, has successfully achieved superior performance in the two performance measures, TT and throughput, when compared to a benchmark algorithm, which does not consider stability in making scheduling decisions.

However, further research is required to confirm whether first-order relationships exist between the stability objective, as defined in this thesis, and both TT and throughput, as well as to determine the type and factors influencing these relationships. Whether stability is more efficient, in optimising the TT and throughput and capturing the tradeoffs, than a multi-objective function is also to be examined.

7.3.1.4 Real Test-bed and Workload

In the evaluation process, the PM-Grid has been deliberately simulated as a logical network to test its design isolated from implementation technologies and platforms, which also conforms with other literature [1, 2]. Additionally, the workload has been synthesised, in conformance with [3-5], to insure flexibility and efficiency in the early stages of development and to provide the basis for cost and time wise evaluation.

However, experimenting with a real test-bed and workloads from a set of different applications is important in early stages of future work to continuously improve the PM-Grid and HoPe designs through feedback arising from real running scenarios.

7.3.1.5 Benchmark Algorithms

Due to the high complexity of the non-clairvoyant scheduling problem, only very few standard heuristics are available [6]. The lack of available information about the system context and resources in PM-Grid environments adds more to the complexity of the non-clairvoyant scheduling problem. Therefore, it was extremely difficult to find and implement a suitable benchmark algorithm for HoPe. Consequently, only the OSH has been utilised to benchmark HoPe. Although the OSH is one of the most often employed and well established heuristics in HTC, contrasting HoPe with other algorithms, such as other bio-inspired heuristics and the round ribbon (RR) algorithm, would help to achieve a more robust evaluation of its performance.

7.3.2 Long Term Future Research

In the long term future research, the following issues are to be explored.

7.3.2.1 PM-Grids

There are several issues that are considered for long term future studies: for instance, trust, privacy and security of users and services in PM-Grids; ethical issues that inevitably arise when sharing personal data or devices; pricing models when a PM-Grid spans multiple PNs or utilises others' resources; resource specification and annotation; service composition and discovery; data management and information services, other job models and task partitioning; fault tolerance and other QoS issues.

7.3.2.2 HoPe

Although the resource scheduling heuristic, HoPe, has been proposed in the context of PM-Grids and specifically for the scheduling problem, by no means it has constraints that limit its application platforms or areas. It could be used in other systems and for other optimisation problems. It may also be generalised to develop a new meta-heuristic for general optimisation problems. Exploring these possibilities is to be considered in future work.

The recent advent of new multi-core processors poses new challenges in developing scheduling algorithms for Operating Systems (OS). Such scheduling algorithms should be designed with adaptability and non-clairvoyant scheduling in mind to cope with the high dynamism in running environments. These features are apparent in HoPe. Therefore, future research intends to explore the possibility of employing HoPe for resource scheduling in multi-core OS as well as other parallel processing systems.

7.3.2.3 Open Issues

There are some open and philosophical issues that are raised by this thesis:

Although grid technologies have never had an explicit goal of changing our society, it is very likely that PM-Grids and other Personal Grids, with the AmI vision as their main driver, will have long-term consequences in our daily lives, as well as ethical concerns that are to a great extent more far-reaching than the Internet.

Finally, it is important to consider that a successful innovation is the result of a specific socio-economic and technological constellation. In other words, the right product, in the right market, at the right time, where specific requirements in terms of user needs, pricing and standards among others, have to be met in order for innovation to succeed and reach its desired objectives [9].

7.4 References

[1] Y. Li, “A bio-inspired adaptive job scheduling mechanism on a computational grid,” *IIJCSNS Int. J. Comput. Sci. Network Security*, vol. 6, no. 3, pp. 1-7, Mar. 2006.

[2] X. Qin and T. Xie, “An availability-aware task scheduling strategy for heterogeneous systems,” *IEEE Trans. Comput,* vol. 57, no. 2, pp. 188-199, Feb. 2008.

[3] R. Buyya, M. Murshed, D. Abramson and S. Venugopal, “Scheduling parameter sweep applications on global grids: A deadline and budget constrained cost time optimization algorithm,” *Softw. Practice Experience*, vol. 35, pp. 491-512, 2005.

[4] C. Dumitrescu, I. Raicu, I. Foster and Di-gruber, “A distributed approach to grid resource brokering,” *in Proc. HPDC-11,* 2002, pp. 352-358.

[5] K. Ranganathan and I. Foster, “Decoupling computation and data scheduling in distributed data-intensive applications,” in *Proc. 11th IEEE HPDC*, 2002, pp. 352- 358.

[6] J. Leung, L. Kelly and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* Boca Raton, FL: CRC Press, 2004.

[7] Jini(TM) Technology [online]. Available: http://www.jini.org/wiki/Jini(TM)_Technology:_An_SOA_Delivering_Java_Dynamic_Networking, [accessed Feb. 24, 2010].

[8] Sun Microsystems [online]. Available: http://www.sun.com/, [accessed Feb. 2, 2010].

[9] I. Miles, K. Flanagan and D. Cox, “Ubiquitous computing: Toward understanding European strengths and weaknesses,” ESTO Report, PREST, Manchester Business School, Manchester, Apr. 2002.

Publications based on this Thesis

Book chapters:

[1] H. Kurdi, M. Li, and H. Al-Raweshidy, “Taxonomy of grid systems,” book chapter, in the *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*, N. Antonopoulos, G. Exarchakos, M. Li and A. Liotta, Eds., USA: Information Science Publishing, 2010.

Journals:

[2] H. Kurdi, M. Li and H. S. Al-Raweshidy, *A Taxonomy of emerging grids*, IEEE Distributed Systems [online], vol.9, no.3, pp.1-13, Mar. 2008. ISSN: 1541-4922

Conferences:

[3] H. Kurdi, M. Li and H. S. Al-Raweshidy, “A bio-inspired heuristic for non-clairvoyant scheduling,” in *Proc. of SiC09,* 5-6 Jun. 2009, Surrey, UK.

[5] H. Kurdi, M. Li and H. S. Al-Raweshidy, “A generic framework for resource scheduling in Personal Mobile Grids based on honeybee colony,” in *Proc. of IEEE NGMAST'08*, pp. 297 – 302. 16-19 Sep. 2008, Cardiff, UK.

[6] H. Kurdi, M. Li and H. S. Al-Raweshidy, “A bio-inspired scheduling heuristic for Personal Mobile Grids,” in *Proc. of ICAC*, Sep. 2008, Uxbridge, UK.

[7] H. Kurdi, M. Li and H. S. Al-Raweshidy, “HoPe: a honeybee inspired resource scheduling heuristic for Personal Mobile Grids,” in *Proc. of SiiC08*, 9-10 Jun. 2008, Leeds, UK

[8] H. Kurdi, H. S. Al-Raweshidy, I. Khan and M. Li, “PM-Grid: a Personal Mobile Grid for ubiquitous computing environments,” Presented at *SiC07,* May 2007, Newcastle, UK.