2009

Scheduling And Resource Management For Complex Systems: From Large-scale Distributed Systems To Very Large Sensor Networks

Chen Yu
University of Central Florida

Scheduling and Resource Management for Complex Systems: From Large-Scale Distributed Systems to Very Large Sensor Networks

by

CHEN YU
B.E. Tsinghua University, 1999
M.E. Tsinghua University, 2002
M.S. University of Central Florida, 2007

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2009

Major Professor: Dan C. Marinescu

ABSTRACT

In this dissertation, we focus on multiple levels of optimized resource management techniques. We first consider a classic resource management problem, namely the scheduling of data-intensive applications. We define the Divisible Load Scheduling (DLS) problem, outline the system model based on the assumption that data staging and all communication with the sites can be done in parallel, and introduce a set of optimal divisible load scheduling algorithms and the related fault-tolerant coordination algorithm. The DLS algorithms introduced in this dissertation exploit parallel communication, consider realistic scenarios regarding the time when heterogeneous computing systems are available, and generate optimal schedules. Performance studies show that these algorithms perform better than divisible load scheduling algorithms based upon sequential communication.

We have developed a self-organization model for resource management in distributed systems consisting of a very large number of sites with excess computing capacity. This self-organization model is inspired by biological metaphors and uses the concept of varying energy levels to express activity and goal satisfaction. The model is applied to Pleiades, a service-oriented architecture based on resource virtualization.

The self-organization model for complex computing and communication systems is applied to Very Large Sensor Networks (VLSNs). An algorithm for self-organization of anonymous sensor nodes called SFSN (Scale-free Sensor Networks) and an algorithm utilizing the Small-worlds principle called SWAS (Small-worlds of Anonymous Sensors) are introduced. The SFSN algorithm is designed for VLSNs consisting of a fairly large number of inexpensive sensors with limited resources. An important feature of the algorithm is the ability to interconnect sensors without an identity, or physical address used by traditional communication and coordination protocols. During the self-organization phase, the collision-free communication channels allowing a sensor to synchronously forward information to the members of its proximity set are established and the communication pattern is followed during the

activity phases. Simulation study shows that the SFSN ensures the scalability, limits the amount of communication and the complexity of coordination. The SWAS algorithm is further improved from SFSN by applying the Small-worlds principle. It is unique in its ability to create a sensor network with a topology approximating small-world networks. Rather than creating shortcuts between pairs of diametrically positioned nodes in a logical ring, we end up with something resembling a double-stranded DNA. By exploiting Small-worlds principle we combine two desirable features of networks, namely high clustering and small path length.

This work is dedicated to my parents, my wife, and my lovely son who have stood by me at all times and believed in me.

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my advisor, Dr. Dan C. Marinescu, for his constant encouragement, steadfast guidance, sound advice, and continues support during each step of my journey toward my Ph.D degree. I have not only been inspired by his insights on academic research, but also been stimulated by his passion and patience in directing my work. Without his instruction and encouragement, this dissertation would have never been possible.

I would also like to thank my committee members, Dr. Charles E. Hughes, Dr. Mark Heinrich, and Dr. Cliff Zou, for their instructions on my research, spending their valuable time reading my dissertation and providing valuable advice and comments.

My thanks also goes to Dr. John P. Morrison, Dr. Ladislau Boloni, and Dr. Gabriela M. Marinescu, for their kind support of my research and helpful comments on my dissertation drafts.

Last but not least, thanks to all the lab mates and all the friends here who accompanied me in the past several years, and thanks to to the departmental staff for their various helps.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In this chapter we provide the motivation for the work reported, and discuss the organization of the dissertation. Then we present an overview of the three major topics covered by this dissertation, discuss the contribution in each area and the related work. The in-depth analysis of each one of the three topics is covered in Chapter 2, 3, and 4, respectively. In each case we present the model and the algorithms, then we report on performance studies and, when feasible, on actual implementations. The systems are generally too complex for analytical performance evaluation and we report simulation results.

1.1 Motivation and Organization

Resource management in a world with finite resources is an enduring problem for social, biological, as well as man-made systems. The term *resource management* is overloaded, it refers not only to resource sharing among entities with conflicting requirements, but also to aspects such as: (i) resource discovery; (ii) protection of resources from unauthorized access; (iii) resource conservation; and (iv) optimal use of resources subject to a set of constraints.

It was recognized early on that a general-purpose computing system should operate under the control of a software component called a Supervisor or Monitor, whose main function is resource management. As computer and communication systems evolved from the ENIAC, to early UNIVAC and IBM systems, time-sharing systems such as Multics, supercomputers such as the CDC and Cray Systems, to personal computers, to clusters, to networks of computers, computational and data Grids, and cloud computing, the problems posed by resource management have evolved. In this evolution new challenges occurred when the resource management was extended from a single system to multiple interconnected systems in the same administrative domain and, finally, when multiple autonomous systems were involved. Autonomy in this case means that systems are in different administrative domains with distinct accounting, security, and resource management policies.

In this dissertation we start with a "classical" resource management problem, namely the scheduling of data-intensive applications. We focus on data-intensive applications because of their importance for science and engineering and because they pose distinct challenges for resource management; they demand massive amounts of resources including CPU cycles, storage, and network bandwidth and require the cooperation of multiple sites. A technological development with potentially significant consequences for scientific and engineering applications is the wide-spread use of many/multi-core processors which may some day lead to the realization of "clusters on a chip." In turn, this will open the possibility of using spare resources of a large numbers of personal computers connected to the Internet via high-speed links for solving data and computationally intensive problems.

A potential solution to resource management in complex systems is self-organization, a strategy which requires systems to cooperate and relinquish for a limited time the total control of resources once a system signs agreements to join an organization. This led us to the second topic discussed in this thesis, a model for self-organization of large-scale distributed systems. Then we apply this model to an architecture called Pleiades. Finally, we discuss the application of self-organization principles to very large sensor networks. We address the problem of resource management at opposite ends of the spectrum: from large-scale distributed systems with abundant resources and a very large population of users sharing these resources to networks consisting of a very large number, 10^8 or more, of primitive devices with very limited resources that have to collaborate to achieve any meaningful task. We hope that the study of such systems will provide new insights and will help us understand the advantages and the problems posed by self-organizing systems.

In this dissertation, we focus on multiple levels of scheduling and effective resource management techniques and present:

- A set of optimal divisible load scheduling algorithms based on a multi-port communication model to achieve the optimal task execution time in heterogenous large-scale distributed computing environment;

- A resource self-organization model based on biological and physics metaphors to optimize the organization of the resources and improve the quality of service in a large-scale heterogeneous distributed computing environment;

- Algorithms for self-organization of anonymous sensor nodes that ensure scalability, limit the amount of communication and the complexity of coordination, reduce energy consumption, and improve the quality of service. We introduce two algorithms, the SFSN (Scale-free Sensor Networks) algorithm and SWAS (Small-Worlds of Anonymous Sensors), an algorithm based on the Small-worlds principle. .

1.2 Divisible Load Scheduling for Data-Intensive Applications

Many applications in computational sciences and engineering such as computational biology [4], computational nanoscience, and many areas of engineering, are data-intensive and computation-intensive. New projects in the emerging field of *e-Science* such as the Teragrid [115], the NAREGI grid in Japan [69], and the ones promoted by the Distributed Geospatial Computing Initiative [64] aim to support data-intensive applications in computer simulations and virtual environments.

Data-intensive and computation-intensive applications demand enormous amounts of resources such as storage space, computing cycles, and communication bandwidth. It is thus natural to distribute a data-intensive computation over a large number of distinct, and often, autonomous systems. The *quantity* of resources reflects also on the *quality* of the infrastructure for data-intensive applications. The processing time for a data-intensive application

is extensive, thus we need effective coordination, an increased level of fault-tolerance, and robust resource selection and scheduling algorithms.

The problem we want to solve is to schedule the execution of independent task instances of a data-intensive application in a heterogenous computing environment consisting of a large number of autonomous systems. Our model for *Divisible Load Scheduling (DLS)* is based on several assumptions: (i) we assume a divisible load, one that can be arbitrarily partitioned; (ii) a Coordinator supervises the data-intensive computation carried out by a large number of Execution Engines; (iii) the task instances running on the Execution Engines process different blocks of data; they are logically equivalent, but their implementations may differ as they are optimized for a particular target system architecture and configuration; (iv) the task instances do not communicate with one another; (v) the performance metric we wish to optimize is the *makespan*, the time from the instant the schedule is generated to the completion time of the last task instance; (vi) the Execution Engines are heterogenous and autonomous systems; thus, the algorithms target a Grid environment.

The divisible load scheduling we consider does not assume any preconditions or dependencies among individual processes. In many cases, the data and control dependencies, as well as the checkpointing, critical for long-running applications, require some form of synchronization among the members of a process group and among different process groups. Different communication patterns lead to different scheduling algorithms. The communication pattern among the communicating processes often includes barrier synchronization, instances when all processes have to exchange some information and thus have to wait for each other before proceeding.

We note that by combining divisible load and co-scheduling techniques, a new coordination strategy called divisible load co-scheduling emerges. We consider a class of applications which consist of two phases of computations. The first phase does not require individual process groups to communicate with each other; the second one can only start after all process groups involved in the first phase have finished. If in the first phase the completion times of

individual process groups are $\{T_1, T_2, \ldots, T_n\}$, respectively, the second phase can only start at $T = \max\{T_1, T_2, \ldots, T_n\}$. The strategy we call *divisible load co-scheduling* guarantees that $T = T_1 = T_2 = \ldots = T_n$, thus it allows the second phase to start at the earliest possible time and all process groups are able to communicate with each other.

Data partitioning for the divisible load co-scheduling strategy is done on two levels [130, 131]. First, the data partitions for the process groups are computed by divisible load co-scheduling algorithms and are distributed to a collection of parallel systems with different resources and startup times; then, when the Phase 1 finishes, the load is partitioned again among processes within one process group based on the local scheduling strategy on each system.

1.3 Complex Systems

The factors affecting the complexity of modern computing and communication systems and implicitly the problems addressed by resource management in such systems are summarized in Figure 1.1. New applications are developed to satisfy the more diverse needs of an increasingly larger population of users. New devices such as sensors find a wide range of applications and, in turn, trigger the development of new applications and attract new users. At the same time, the individual components of a system are more complex, for example, 2-4 core processors are ubiquitous now and will be replaced by multi-core systems with tens to hundreds of cores in the next few years. Connectivity and mobility, required by virtually all users of the systems, increase the complexity of the applications and of the computer and communication infrastructure. At the same time, physical limitations, such as heat removal, bandwidth availability, the capacity to store energy, and other factors place additional constrains upon system design. Last, but not least, we have finite resources and the optimization of resource consumption increases system complexity.

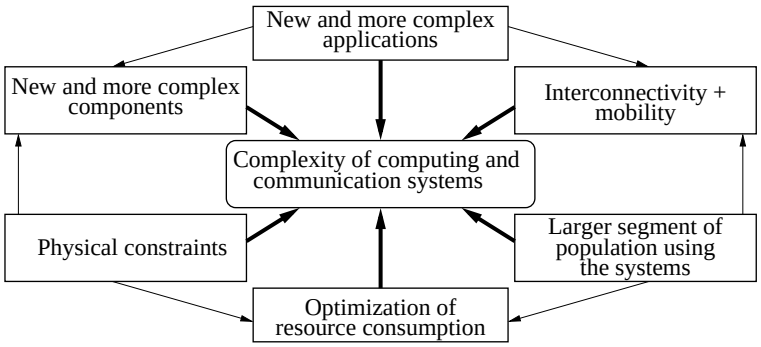


Figure 1.1: Some of the factors contributing to the complexity of modern computer and communication systems.

The computer and communication systems appear increasingly more complex to their users and this has triggered the investigation of metrics for the *complexity of use of distributed systems.* Some of these metrics are [95]: (i) task-structure complexity, how difficult it is to understand how to perform a task; (ii) unpredictability, how difficult it is to predict the effects of an action; (iii) size complexity; (iv) chaotic complexity, reflects the fact that small variations in a certain part of the system can have large effects on overall system behavior; and (v) algorithmic complexity.

The very practical question on how to design, maintain, and use complex computing and communication systems cannot be answered without some understanding of the general characteristics of complex systems, the metrics that allow us to assess the complexity of a system, and without some insight into how complex systems in nature behave. Some of these considerations lead to very abstract questions that have preoccupied the minds of humans for millennia. For example, Aristotle stated that "...the whole is something over and above its parts, and not just the sum of them all.." In "The Republic" Plato, introduces the concept of "level of knowledge" ranging from total ignorance to total knowledge. True knowledge exists only if a foundation of axioms or a priori knowledge exists [50] and this cannot be the case for complex systems.

While we have an intuitive notion of what complexity means, a rigorous definition allowing us to quantify and measure the complexity of a system is not universally accepted. Certainly, the scale of a system, the dynamics of the system behavior, the unpredictability of the next state, the time a system has been around may affect its complexity, but as we shall see none of these elements by itself allows us to conclude that a system is complex or not.

The thermodynamic entropy, von Neumann entropy, and Shannon entropy are related to the number of states of a system, thus they reflect to some extent the system complexity [30]. A measure of complexity introduced by Crutchfield is the *relative predictive efficiency*, $e = E/C$ with E the excess entropy and C the statistical complexity [31]. The *excess entropy* measures the complexity of the stochastic process and can be regarded as the fraction of historical information about the process that allows us to predict the future behavior of the process. The *statistical complexity* reflects the size of the model of the system at a certain level of abstraction. The scale of organization considered by an external observer plays a critical role in assessing the relative predictive efficiency. For example, at the microscopic level the calculation of e for a volume of gas requires very complex molecular dynamics computations to accurately predict the excess entropy; both E and C are very high and the predictive efficiency is low. On the other hand, at the macroscopic level the relationship between the pressure P, the volume V, the temperature T is very simple $PV = nRT$ with n the number of moles of gas and R the universal gas constant. In this case E maintains a high value, but now C is low and the predictive efficiency E/C is large.

Kolmogorov entropy [61,66] of an object is a measure of computational resources needed to specify the object. Using a Markovian assumption and a Kolmogorov complexity model, it was shown that scheduling and resource management are more complex on a computational grid than on a service grid due to the finer granularity of resource allocation [71].

We propose to use the complexity of a program that simulates the system as a measure of complexity of the system; this will reflect not only the number of states but also the

transitions among states. This proposal has its own limitations, as we generally simulate approximate models of a system, rather than exact ones. The proposal is consistent with the concept of *depth* defined as the number of computational steps needed to simulate a system's state [70]. Machta argues that the emergence of complexity requires a long history, but one needs a measure stricter than physical time to reflect this history [70]. The depth reflects not how long the system remains in equilibrium, but *how many steps are necessary to reach equilibrium following some efficient process.* The rate of system state changes and the communication time do not reflect the complexity of a system. Indeed, two rotating structures involving very different physical processes, a hurricane and a spiral galaxy are at the limit of today's realistic computer simulation thus, of similar depth and, consequently, of similar complexity. Yet, galaxy formation occurs at a scale of million light years and it is bounded by communication at the speed of light, while the time for hurricane formation is measured in days, the atmospheric disturbances propagate more slowly, but the the scale of hurricane formation is only hundreds of kilometers.

Physical systems in equilibrium display their most complex behavior at *critical points* (in thermodynamics a critical point specifies the conditions, temperature and pressure, at which a phase boundary, e.g., between liquid and gas, ceases to exist). The time to reach equilibrium becomes very high at critical points, a phenomena called *critical slowing.* Wolpert and Macready [117] argue that *self-similarity* can be used to quantify complexity; the patterns exhibited by complex systems at different scales are very different, while the patterns exhibited by simple systems such as gases and crystals do not vary significantly from one scale to another.

The two most important concepts for understanding complex systems are emergence and self-organization. *Emergence* lacks a clear and widely accepted definition, it is generally understood as *a property of a system that is not predictable from the properties of individual system components.* There is a continuum of emergence spanning multiple scales of organization. Halley and Winkler argue that simple emergence occurs in systems at, or

near thermodynamic equilibrium while complex emergence occurs only in non-linear systems driven far from equilibrium by the input of matter or energy [44].

1.4 Self-organization

The term "self-organization" is used in the literature in several contexts; Minsky [83] has distinguished the informal from the technical use of the term "self-organization," while Gell-Mann [40] has questioned the usefulness of the technical use of it without precisely defining concepts such as levels of organizations, global and local information, and patterns of behavior [29]. Though the concept of self-organization is difficult to define [81] its intuitive meaning is reflected by the observation of Alan Turing that "global order can arise from local interactions" [108]. In his seminal paper on morphogensis Turing suggests that a mechanism based upon slow diffusion of an activator and fast diffusion of an inhibitor can lead to *formation of stable stationary non-equilibrium patterns through the diffusion of some key compounds.* There is no straightforward path from this idea to the design principles for self-organizing computing and communication systems. Known techniques for solving optimization problems such as genetic algorithms or simulated annealing cannot be applied directly to self-organization when we wish to reach stable stationary non-equilibrium patterns.

Inspired by biological systems, self-organization was proposed for networking [81] and even for economical systems [62]. Self-organization of biological systems is defined as "a process in which patterns at the global level of a system emerge solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the systems's components are executed only with local information, without reference to global patterns" [24]. In this dissertation we adopt the view that a system is self-organizing if its components interact to dynamically achieve a global function or behavior. Self-organizing systems have higher level properties that cannot be observed at the

level of the individual components and that can be seen as a product of their interactions, they are more than the sum of their parts.

Self-organization is used by different flavors of neural networks including Hopfield networks [47], or the networks proposed in [81]. The "swarm" algorithms [20], e.g., the Ant Colony Routing, mimic self-organization of social insects. Self-organization schemes have been proposed for *ad-hoc* [110] and sensor networks [29, 75]. The current literature on self-organization provides theoretical insights, but few clues on how to apply the basic principles for systems design [42, 46]. Indeed, this is a daunting task due to the wide spectrum of potential applications with dissimilar physical constrains and objectives.

Several ad-hoc solutions for managing complexity have been proposed. The autonomic computing initiative [28,45] is motivated partially by the fact that advances in scale and complexity of modern systems outpace the ability to configure, maintain, and run these systems. An autonomic system is composed of components expected to exhibit self-configurability, self-healing, self-optimization, and self-protection. The components of an autonomic system are required to be self-aware and contextually aware (knowledge of itself, its environment, and the environment that it is executing in), open (to support the wide range of different components that could be incorporated into the system), and anticipatory (to anticipate resources needed). These desiderates, though consistent with the self-organization principles, did not lead to the development of a coherent autonomic computing model.

Two other approaches to managing the complexity of modern computer and communication systems are based upon economic models and self-organization. Economic models and financial markets offer an alternative perspective on the management of complex systems. The on-going effort to apply concepts from financial markets to large-scale distributed computing, and to grid computing in particular, has generated a number of research ideas [6, 77, 98, 102]. However, financial markets concepts have not been applied in practice, though it is agreed that computing and communication resources such as CPU cycles and communication bandwidth are perishable commodities that can be traded.

Table 1.1: Self-organization and complexity

Simple systems lacking self-organization	Complex systems exhibiting self-organization
Mostly linear	Non-linear
Close to equilibrium	Far from equilibrium
Tractable at component level	Intractable at component level
One or few scales of organization	Many scales of organization
Similar patterns at different scales	Very different patterns at different scales
Do not require a long history	Require a long history
Simple emergence	Complex emergence
Limited scalability	Scale-free

A market-oriented approach to resource management seems antithetic to self-organization; the former is a product of the human mind and reflects societal and economic views, while the latter is inspired by nature and attempts to mirror the behavior of biological systems. Markets assume *global* knowledge and, occasionally, the intervention of a controlling authority, such as the Federal Reserve Board or the European Central Bank, while the entities of a self-organizing system act primarily based on *local* information. Moreover, agents operating in a market environment *react* to the market, while the entities of a self-organizing system *act on their own initiative.* In the case of the markets, a self-stabilizing mechanism does not exist, thus, the need for the controlling authority mentioned above; in self-organizing systems the information spreads relatively slowly thus, the danger of catastrophic events caused by panic is limited.

Our limited understanding of system complexity and the highly abstract concepts developed in the context of natural sciences do not lend themselves to design principles for modern computing and communication systems. Nevertheless, the generic attributes of complex systems exhibiting self-organization, summarized in Table 1.4, allow us to identify some of the specific factors affecting the complexity of these systems [1]:

- The physical nature and the physical properties of computing and communication systems must be well understood and the system design must obey the laws of physics.

- The behavior of the systems is controlled by phenomena that occur at multiple scales/levels. As levels form or disintegrate, phase transitions and/or chaotic phenomena may occur.
- The systems have no predefined bottom level; it is never known when a lower level phenomena will affect how the system works.
- Abstractions of the system useful for a particular aspect of the design may have unwanted consequences at another layer.
- The systems are entangled with their environment. A system depends upon its environment for its persistence, therefore it is far from equilibrium. The environment is man made and the selection required by the evolution can either result in innovation, generate unintended consequences, or both.
- The systems are expected to function simultaneously individually and as groups of systems (systems of systems).
- Typically, computing and communication systems are both deployed and under development at the same time.

The question "where can the boundary between abstract and practical knowledge about complex systems be drawn?" led us to the development of the generic model introduced in [77] and expanded in the next sections where we introduce a generic model by applying concepts from natural sciences for self-organization.

1.5 Self-organization, Large-scale Distributed Systems and the Pleiades

We develop a self-organization model for resource management in large-scale distributed systems consisting of a very large number of sites with excess computing capacity. Based on the current trends in microprocessor, memory, and disk technology it seems reasonable to expect that the personal computer of the future will be a multi-core system with tens,

if not hundreds, of cores, Gbytes of cache, tens of Gbytes of main memory and hundreds of Gbytes, if not Terabytes, of secondary storage. If a secure and reliable infrastructure supporting resource virtualization is in place then one could expect that the owners of such systems will be motivated to trade their excess computing capacity for a variety of incentives such as monetary rewards, service credits, and software maintenance contracts.

Pleiades [1] is a proposed service-oriented architecture based on resource virtualization [76]. This architecture supports an infrastructure enabling resource-starved service providers to gather and use computing resources from a very large number of systems connected to the Internet via high-speed links. The resource providers could also be clients and users of services.

The actual realization of such an architecture requires a paradigm shift in resource management and system security. In [76] we propose a hierarchy of virtual machines (VMs) to run on each system involved. The top VM supports negotiations with the outside world and enforces a strict division of the system into a *host* and a *guest* partition; a partition consists of cores, disk space, network bandwidth, and other resources and it is isolated from the other. Once negotiated, such a division is enforced until the contract to provide resources to the outside world is fulfilled; breaking a contract incurs severe penalties. To be efficient, the contracts should cover a reasonable time, minutes or tens of minutes. The second level VMs manage the two partitions: one VM controls the host partition and supports multiple operating systems; the other VM controls the guest partition and allows services to be installed at the request of service providers. Such a strategy guaranties a stable supply of resources for service providers.

In this dissertation, we outline a self-organization model and apply it to study a Pleiades-type system.

[1] The Pleiades (Messier object 45) are an open star cluster in the constellation of Taurus. The cluster core radius is about eight light-years; the cluster contains over 1,000 statistically confirmed members and the total mass contained in the cluster is estimated to be about 800 solar masses.

1.6 Communication Scheduling in Self-organizing Very Large Sensor Networks (VLSNs)

A sensor network consists of spatially distributed autonomous devices equipped with a radio transceiver or other wireless communication device, a micro-controller, a power source, and sensors that monitor temperature, sound, vibration, pressure, motion, chemical pollutants, radiation, or other physical characteristics of the environment. The application software on the nodes of a sensor network typically uses a network stack communicating with a middleware layer running on top of a real-time operating system.

We consider networks consisting of a very large number of tiny and inexpensive sensors. The nodes of Very Large Sensor Networks (VLSN) have limited resources; to reduce the power consumption the processor is less powerful and the amount of storage available is smaller than those of traditional sensor networks. Moreover, the nodes are indistinguishable from one another; they do not have a physical address, as required by the traditional communication protocols and mimic biological systems where individual cells of the same type are indistinguishable.

Deployment of such systems with a high density of sensors over a large geographic area could be used to monitor the environment and study climate changes; the system could produce a histogram used to calculate the moments of the distribution of one or more physical or chemical properties of the environment monitored by the sensors. For example, when the area to be monitored is 10 mile^2, and the density is 4 sensors/10 ft^2, the total number of sensors is about 10^8.

Typical applications could be studies of the temperature of the polar cap, of the glaciers in the Alps, of the sol humidity in a forest, or of the tremors in an earthquake prone region. The sensors should be able to operate with their power reserves for a year or more. The maintenance, as well as the operation of the network, must be inexpensive. The sensors are dropped from an aircraft or planted by a specialized device and are stationary or experience

very limited mobility; an unmanned aircraft flies over the area and collects the information at pre-determined time intervals.

In this dissertation, we present communication scheduling for two self-organization algorithms for the VLSNs: Scale-free Sensor Network (SFSN) and Small-worlds of Anonymous Sensors (SWAS).

1.6.1 Scale-free Sensor Networks (SFSN)

An important attribute for self-organizing systems is scalability, the ability of the system to grow without affecting its global function(s). Complex systems encountered in nature, or man-made, exhibit an intriguing property, they enjoy a *scale-free organization* [8, 9]. We believe that this property reflects one of the few attributes of self-organization that can be precisely quantified. The scale-free organization can be best explained in terms of the network model of the system, a random graph with vertices representing the entities and the links representing the relationships among them. In a scale-free organization the probability $P(m)$ that a vertex interacts with m other vertices decays as a power law, $P(m) \approx m^{-\gamma}$, with γ a constant, regardless of the type and function of the system, the identity of its constituents, and the relationships between them.

Empirical data available for power grids, the Web, social networks, or the citation of scientific papers, confirm this trend; the systems discussed in [8, 9] are complex and based upon accumulated knowledge of past behavior. An interesting question is if scale-free, self-organizing *primitive* systems like sensors which have limited resources and cannot store historical information, receive, process, or transmit large volumes of data, exhibit notable advantages over more traditional organizations of sensor networks.

The SFSN algorithm is based upon the idea that a sensor should only maintain information about a very small number of sensors in its proximity and about a small number of events when it is expected to wake up and transmit or receive data. The algorithm shares some

ideas with the Self-organizing Medium Access Control for Sensor Networks (SMACS) [99]. SFSN and SMACS first determine the radio connectivity in the network and then assign collision-free channels to the links; both assume limited mobility. Unlike the Link Clustering Algorithm in [7] which performs two passes, the first carried over the entire network to discover neighbors and the second to assign channels to links between two neighbors, SFSN and SMACS assign immediately a channel to a link. Other similarities: both assume that nodes are able to turn their radio on and off; the nodes are able to tune the carrier frequency to different bands and the number of available bands is quite large; the nodes form a flat topology rather than clusters.

The major differences between SFSN, the algorithms we propose, and SMACS [99] are:

(i) The nodes of a SFSN network are anonymous, they do not have either a physical or a logical address;

(ii) A node of a SFSN network communicates using multiple unidirectional channels;

(iii) For SFSN the reorganization of the network occurs periodically, while for SMACS there is only one setup phase followed by a steady-state operation mode;

(iv) The virtual channels assigned during the self-organization phase in SFSN network are implicit and related only to the index of the communication event, the nodes do not exchange a schedule for communication. In SMACS networks the assignation of the channels is explicit, the carrier frequencies are chosen once and for all and two nodes exchange the schedule of transmissions for the entire duration of the steady-state operation;

(v) Multiple nodes may choose to respond to an invitation to transmit during the self-organization phase in SFSN and collisions are likely; though collisions may occur during the assignation of the channel in SMACS, these are rare events;

(vi) For SFSN the self-organization phase proceeds strictly sequentially, thus, the setup phase may take longer; an extension of the basic algorithm allows concurrent setup. The network

setup can be done in parallel in SMACS, multiple nodes may initiate the assignment of the channel at the same time.

1.6.2 Small-worlds of Anonymous Sensors (SWAS)

The small-world phenomenon known also as "six degrees of separation" reflects the fact that we are all linked by short chains of acquaintances. The question we addressed is if this principle could be applied to create sensor networks with several properties: (i) they are not in danger to be disconnected when sensors fail; (ii) they have a relatively short average path length; (iii) they minimize the power consumption to communicate, in other words establish collision-free communication channels as well as a schedule that minimizes the time when each has to wake up and transmit or receive; (iv) each sensor maintains a very limited amount of information.

Traditionally, the connection topology of a network was assumed to be either completely regular, or completely random. Regular graphs are highly clustered and have large characteristic path length, while random graphs exhibit low clustering and have small characteristic path length. The *characteristic path length,* L, is the number of edges in the shortest path between two vertices averaged over all pairs of vertices. The *clustering coefficient* C is defined as follows: if vertex a has m_a neighbors, then a fully connected network of its neighbors could have at most $E_a = m_a(m_a - 1)/2$ edges. Call C_a the fraction of E_a of edges that actually exist; C is the average of C_a over all vertices. Clearly, C measures the degree of clusterings of the network.

In 1998, D. Watts and S. H. Strogatz studied the graphs combining the two desirable features, high clustering and small path length, and introduced the Watts-Strogatz graphs [113]. They proposed the following procedure to interpolate between regular and random graphs: starting from a ring lattice with n vertices and m edges per node rewire each edge at random with probability $0 \leq p \leq 1$; when $p = 0$ the graph is regular and when $p = 1$

the graph is random. When $0 < p < 1$ the structural properties of the graph are quantified by: (i) the characteristic path length, $L(p)$, and (ii) the clustering coefficient, $C(p)$. If the condition $n >> m >> \ln(n) >> 1$ is satisfied then $p \to 0$ leads to $L_{regular} \approx n/2m >> 1$ and $C_{regular} \approx 3/4$, while $p \to 1$ leads to $L_{random} \approx \ln(n)/\ln(m)$ and $C_{random} \approx m/n << 1$.

The small-worlds networks have many vertices with sparse connections, but are not in danger of getting disconnected; moreover, there is a broad range of the probability p such that $L(p) \approx L_{random}$ and, at the same time $C(p) >> C_{random}$. The significant drop of $L(p)$ is caused by the introduction of a few shortcuts which connect vertices that otherwise would be much further apart. For small p, the addition of a shortcut has a highly nonlinear effect; it affects not only the distance between the pair of vertices it connects, but also the distance between their neighbors. If the shortcut replaces an edge in a clustered neighborhood, $C(p)$ remains practically unchanged, as it is a linear function of m.

Self-organizing small-worlds networks pose a fair number of challenges. It is non-trivial to apply Watts-Strogatz ideas to construct a regular graph, e.g., a ring, and then to create shortcuts among distant nodes in the logical network. The problems are even more difficult, when the nodes are indistinguishable and have a limited ability to store state information. The SWAS algorithm ensures scalability, as the number of nodes each sensor communicates with and the amount of state information each node has to maintain are strictly limited, regardless of the total number of sensors in the network. This scheme limits the amount of communication and the complexity of coordination.

1.7 Contributions

The work we report is tied to our effort to build an intelligent resource management environment for the complex systems scaled from large-scale distributed systems to the very-large sensor networks. The contribution of our work includes:

- A set of optimal divisible load scheduling algorithms for data-intensive applications in a heterogeneous large-scale distributed environment that exploit parallel communication, consider realistic scenarios regarding the time when target systems are available, and generate optimal schedules; the simulation shows the model presented outperforms the corresponding models with one-port communication;

- A self-organization model based on biological and Physics metaphors for resource management in large-scale distributed computing environment; the algorithm for self-organization is introduced with the simulation that shows that the presented model efficiently reduces the resource consumption and increases the quality of service.

- An algorithm for self-organization of anonymous sensor nodes called SFSN (Scale-free Sensor Networks) and an algorithm utilizing the Small-worlds principle called SWAS (Small-worlds of Anonymous Sensors). The self-organization scheme we propose ensures scalability, the number of sensors each sensor communicates with and the amount of state information each node has to maintain are strictly limited regardless of the total number of sensors in the network; a node in the random graph showing the system's connectivity is linked with at most a pre-determined number of nodes, thus the system is scale-free. The application of Small-worlds principle further combines two desirable features of networks namely high clustering and small path length. These schemes limit the amount of communication and the complexity of coordination.

CHAPTER 2: DIVISIBLE LOAD SCHEDULING ALGORITHMS BASED ON THE MULTI-PORT COMMUNICATION MODEL

In this section, we introduce the divisible load scheduling model, present the optimal divisible load scheduling algorithms, report performance evaluation results, and outline the design of a divisible load scheduling meta-scheduler.

2.1 Divisible Load Scheduling

Scheduling policies for heterogeneous and autonomous systems [37] consider the service availability, computation power, network characteristics, and the workload of individual systems. Scheduling decisions are made locally and not under the control of a single authority. Space sharing of a single cluster ensures that all nodes available start processing at the same time but this is no longer true in a Grid environment where individual systems become available at different times. Thus, the classic data partitioning and scheduling algorithms for the SPMD model [32, 89, 114] in the dedicated parallel environment as well as the co-scheduling algorithms [5, 48, 56] for the heterogeneous computing platform cannot be efficiently exploited as the preemptive condition that the resources are simultaneously allocated to all the various parallelized processes is difficult to achieve. In addition, the remote input data needs to be fetched to the Grid resource before the computation starts, a process called *data staging*. The relatively low speed network bandwidth and high latencies may lead to substantial performance degradation in a Grid environment and impose notable penalties [79]. Hence, the scheduling algorithms that only take into account the computation power of the resources and ignore the network transfer costs are unlikely to be efficient. Many works such as [10, 82] incorporate the data transfer cost into the scheduling process with the assumption that the *expected data transfer cost* (communication cost) is known or can be estimated beforehand. On the contrary, our work is more sophisticated as we have no knowledge of the expected

data transfer cost to be provided to the scheduling algorithm as input. It becomes clear only after the schedule is generated and the data partitions are computed.

The study of the Divisible Load Theory (DLT) was initiated in 1996 [13] and a comprehensive summary of the research prior to 2003 can be found in [14]. DLT is a linear mathematical model and provides a practical framework for the mapping of the independent task instances onto heterogeneous computing platforms. Applications of image processing [63], biological computing [54, 65], multimedia applications [3], or database record searching [15] can be approached as divisible load. Initially one-round algorithms that allocate exactly one block of data to each system were studied for several network topologies: linear [26], star) [12], multi-level tree [27], bus [11], hypercube [16] and mesh [17]. Multiple-round algorithms were introduced in [125]; in this case a pipelined execution was carried out; a fraction of the load assigned was processed while receiving the next.

Minimization of the makespan of an application with multiple independent tasks on heterogenous environments is discussed in the literature [10, 22, 67, 96, 106]. For example, in [22], the authors compare eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems while in this dissertation we discuss a single data intensive computation rather than trying to optimizing the performance of multiple tasks. Although the problems seem similar, there are subtle differences making it impossible to apply the heuristics of [22] to our problem. First, we have no a priori knowledge of the *expected execution time* (computation and communication cost) for each task instance; in our case the partitioning of the data across heterogeneous systems and the scheduling are intimately correlated. The execution time is directly related to the workload allocated to each task instance which needs to be computed by the scheduling algorithm. The scheduling algorithm for the DLS problem not only generates the optimal schedule but also calculates the data assigned to each task instance. The number of task instances can either be statically (e.g., required by application) or dynamically produced by the DLS scheduling algorithm.

Several papers discuss the application of the DLT to Grid computing [60, 84, 92, 109, 120, 132, 133]. The DLT strategies for the star-shaped and tree-shaped topologies are studied in [12]. In the one-port model the coordinator site cannot distribute the workload to execution engines in parallel, it can only communicate with a single execution engine at a time. Both the one-round and multi-round algorithms are discussed together with the analysis of optimality of the solution. Despite the similarity of the platform and application model, there are a number of important differences between previously reported work e.g., [12], and the work reported in this dissertation. First, we assume a multi-port communications model which enables the coordinator to distribute the load to execution engines in parallel. For reasons of performance and data security, data-intensive tasks store the data at "data centers" with massive communication bandwidth and low communication latencies. New network technologies substantially reduce the overhead of parallel data transfer and we expect that multi-port communications will become increasingly more popular. Second, we introduce a new data staging strategy different from one-round and multi-round strategies; we assume the data staging can begin after the schedule is generated and terminate before the scheduled computation start up time on an execution engine. It has been proved that all execution engines should finish computing at the same time to achieve the optimal solution for one-round and multi-round strategies [12,121], but we will show this is not always true for the data staging strategy we just defined. Last but not least, resource selection is included in the DLS algorithms. In contrast to the linear case where all available execution engines participate we select a subset of resources from the resource pool for parallel execution.

2.2 Basic Concepts

A *process group* $\mathcal{G} = \{P_1, P_2, \ldots, P_n\}$ is a set of n processes running concurrently on a set of cores of a multi-core system, on the nodes of a cluster, or on several systems interconnected by a high-speed network. Scheduling a process group in the context of this dissertation means

the ability to first select a subset of target systems from a pool of potential candidates, subject to the constraints regarding the resources available on each system (including the existence of binaries for a specific application) and the time when they are available, compute an optimal load distribution, and finally start the computation on each system. The first aspect of scheduling is known as *mapping*; the second is *load balancing*. Starting the computations requires interactions with the host operating system and involves some form of pre- and post-processing including data staging which is the process of transfering the workload to a site prior to the start of the computation.

As multi-core systems become ubiquitous, more and more applications require a large amount of computing cycles and it makes sense to consider multiple process groups, one per multi-core system, running concurrently. Concurrent execution on multiple systems requires some form of coordination [18]. The coordination algorithm is executed by a workflow enactment engine [68] that supervises not only the execution of the computation, but also the pre- and post-processing activities including data staging and exception handling, e.g., the failure of one or more target systems. In the general case the systems we target for execution are autonomous, they belong to different administrative domains, and scheduling requires an agreement among them.

Several factors contribute to making the effective scheduling and coordination strategies a hard problem:

- Heterogeneity of the target systems. The parallel machines that comprise the distributed system may differ in the number of nodes, main memory and cache per node, secondary storage, communication bandwidth, etc.

- Autonomy of the individual systems usually located in different administrative domains. Scheduling decisions are made locally and not under the control of a single authority. Indeed, space sharing of a single cluster ensures that all nodes available for

a given computation start processing at the same time, while this is no longer true for multiple clusters that may become available at different times.

- High data staging costs among systems. Data staging for massive amounts of data requires substantial communication bandwidth and takes a considerable amount of time.

- Reliability considerations. We have to maintain a considerable amount of state information to be able to recover from possible faults.

- The need to coordinate execution among several systems. The larger the number of systems involved, the larger the overhead for coordination and the probability of failure.

Co-scheduling is a technique to schedule all processes in a process group for execution at the same time; when a process in the group blocks while communicating with another process in the group, the local scheduler leaves its state loaded on the processor for a short time, under the assumption that it will receive a response shortly [5]. Co-scheduling of process groups had been investigated since early 1990's when networks of workstations were used for parallel computing [5, 48]. The objective of co-scheduling is to finish the computation at the earliest time. Thus, the local schedulers of the autonomous systems must reach an agreement regarding the earliest time when all systems we target are available.

The divisible load scheduling strategy we introduce in this dissertation can be combined with co-scheduling techniques and thus leads to a different communication pattern. We consider a class of applications that share the following characteristics: the computations consist of two phase; the first phase does not require individual process groups to communicate with one another; the second one can only start after all process groups involved in the first phase have finished. If the completion time for the first phase of individual process groups are $\{T_1, T_2, \ldots, T_n\}$, respectively, then the second phase can only start at $T = \max\{T_1, T_2, \ldots, T_n\}$. The divisible load strategy we introduce guarantees that

$T = T_1 = T_2 = \ldots = T_n$, thus it allows the second phase to start at the earliest possible time and all process groups are able to communicate with one another. Thus, the data partitioning for this two phased execution model is done at two levels [128, 130, 131]; first, the data partitions for the process groups are computed by divisible load scheduling algorithms and are distributed to a collection of parallel systems with different resources and available times; then, when the Phase 1 finishes, the load is partitioned again among processes within one process group based on the local scheduling strategy on each system.

Figure 2.1 illustrates graphically the timing constrains of the execution model combining the divisible load scheduling and co-scheduling and of the traditional co-scheduling models; in the first case, process groups start Phase 1 at different times but finish it at the same time, thus are able to start Phase 2 at the same time. In case of traditional co-scheduling all process groups must start Phase 1 at the same time.

Any co-scheduling algorithm satisfies the condition that all parallel processes or process groups terminate computation at the same time, but will not guarantee optimality, the earliest possible completion time. The need to use multiple multi-core systems is predicated upon the fact that such applications require massive amounts of CPU cycles; moreover, many applications are data-intensive and using a limited number of cores on each system may alleviate the I/O bottleneck problem. While the local scheduler of a multi-core system may be able to guarantee that all processes of the process group $\mathcal{G}_i$ can start at the same time, there is no guarantee that two process groups $\mathcal{G}_i$ and $\mathcal{G}_j$ running on two different systems will be able to start at the same time. This may exclude some potential target systems even if they have the superior computation power, the system E in figure 2.1 for example, and will delay the computation startup time for the Phase 1. Indeed, if the n systems selected by the co-scheduling algorithm from a pool of N systems lead to a completion time T for the first phase, than any one of the remaining $N - n$ systems available at a time before T could be added to the set of previously selected systems and thus, the first phase of the computation could finish at a time earlier than T.

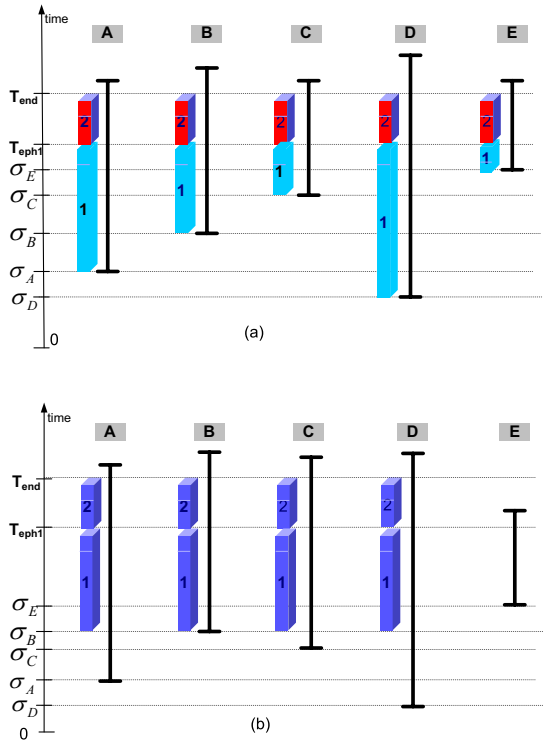


Figure 2.1: (a) Divisible load scheduling combined with co-scheduling. The solid bars show the intervals when the five systems labeled A, B, C, D and E are available starting at times $\sigma_A, \sigma_B, \sigma_C, \sigma_D$ and σ_E, respectively. The lower blocks correspond to phase 1 when processing starts as soon as a system becomes available, the upper blocks correspond to phase 2; all systems finish phase 1 at the same time, T_{eph1}, start phase 2 at the same time, and finish phase 2 at the same time, T_{end}. (b) Co-scheduling. All processes start phase 1 at the same time, $\sigma_B = \max(\sigma_A, \sigma_B, \sigma_C, \sigma_D)$; all finish phase 1 at the same time, T_{eph1} and phase 2 at T_{end}; E is not chosen because it would delay the start-up time of phase 1 and it is not available for the entire duration of the execution of phase 2.

An important class of scientific applications that fit this coordination pattern is the Monte Carlo (MC) simulation used in nanoscience, physics, chemistry, and many other areas of computational sciences. For example, virtually all simulators of fault-tolerant quantum circuits use MC simulation and the error threshold for a particular set of quantum gates and error correcting codes is given by the ratio of the number of runs when the system "crashed" to the total number of runs. In this case, the first phase is the actual MC simulation and the second phase covers the reliability analysis. There are other computational science applications in the same class. Our interest in the divisible load co-scheduling was motivated

by a real-life application for virus structure determination [54] which uses a parallel algorithm for origin and orientation refinement for the first phase of an iteration and then a 3D-reconstruction algorithm for the second phase. To reconstruct a 3D electron density map of a virus, we use a number of 2D projections from micrographs obtained experimentally with a cryo transmission electron microscope. The computing time required to improve the resolution of a medium-sized virus such as the Mammalian Reovirus (MRV), from about 7.6 Å to better than 7.0 Å on 42 processing nodes of one cluster, is about 14 hours/iteration. The refinement process requires about 100 iterations, thus the total time is about $1,400$ hours, or nearly 60 days. Such sobering statistics show the benefit of using concurrently multiple parallel systems.

2.3 Problem Formulation

In this section we discuss the characterization of resources, the model of the system, and data staging strategies.

2.3.1 Characterization of Resources

We discuss four elements used in our quantitative analysis of the system: the execution rate, the duty cycle, the available time, and the data transfer rate.

2.3.1.1 Execution Rate

Different measurements of the bandwidth of a computing engine have been defined in the literature. For example, in [10], the MFLOP (mega floating-point operations per second) is used to quantify the task's computation requirement and the resource's computational capability. Another commonly used measurement is MIPS (million instructions per second), which indicates integer operation performance.

Each of these measurements focus on one aspect of the computation power, but we argue that the computational capability should be a synergistic measure of the computation power that reflects a wide range of computational resource attributes, such as the system architecture, the CPU clock rate, the memory access time, the amount and speed of memory cache, and the I/O latency and bandwidth. The comprehensive consideration of these attributes is significant for a reasonable estimate of the computational capability of a resource on a given data intensive task which may involve intensive I/O operations or memory accesses leading to substantial performance degradation.

Assuming the input data is available at the computation resource where the execution engine resides, the execution of a data-intensive task involving the large-scale input data exhibits the behavior that the data processing time dominates the total computation time which includes the program start up, data processing and program termination cost. As a result, it is a reasonable approximation to represent the computational capability by the amount of input data processed per unit of time.

In this thesis we measure the size of input data set, denoted by ω, for a data intensive task by *data unit (dtu)*, which reflects the logical organization of the data for the task. For example, for the virus origin and orientation refinement application [54], a *dtu* consists of a number of virus projections extracted from a micrograph. We define the *execution rate* μ_i of a task on resource i as the amount of input data, locally available on resource i, processed in one unit of time when resource i is fully loaded with this task.

It is challenging to practically measure or predict the execution rate of a data-intensive computation on a resource. One approach to estimate the execution rate is to run the code with sample input on the target resource. With less accuracy, the static estimation can also be made by comparing the resource attributes of a new resource to the others with known execution rate. The resource attributes may show different weights when making the comparison, accounting for the characteristics of the task. In addition, although we use the abstract concept, the data unit, to measure the problem size, the scheduling algorithms

presented in the following section work on the other measurements as well, MFLOP/MIPS for instance.

2.3.1.2 Duty Cycle

A computing engine can be time-shared or space-shared depending on the machine architecture or operating system. Time-sharing refers to multitasking, while a multiprocessor system can be space-shared by allocating processors to multiple tasks running concurrently. The resource sharing strategy is controlled by the local scheduler or vendor supplied operating system. Determining the practical execution rate of a task on a resource requires a prediction of the computation power allocation either implicit or explicit.

We do not differentiate between two resource sharing models above and define the *duty cycle* η_i as the fraction of the computation power available for the given task on the resource i. With the execution rate μ_i and the duty cycle η_i, the practical execution rate of the task on resource i will be: $\mu_i\eta_i$.

The sharing of the computation power reflects the resource allocation on a wide rage of resource attributes: the CPU clock rate, the I/O bandwidth, the memory access bandwidth and so on, which makes it complicated to measure and predict. Although the accurate measurement can be done by running the *test process*, a reasonable approximation to the duty cycle in most cases is the *CPU availability*. For the time-sharing system, *CPU availability* is defined to quantify the fraction of the CPU that can be exploited by a given task [23, 97]. In [118], the authors discussed a model to predict the CPU availability on the time-shared Unix systems. In the case of space-sharing system, the CPU availability is the fraction of CPUs that will be allocated to the given task, a decision made by the local scheduler or vendor supplied operating system.

2.3.1.3 Available Time

The local scheduler on a resource maintains the job queues and the time when a job is scheduled to start the execution can be estimated. The estimation is explicit with the knowledge of jobs' submission description files, the resource allocation strategy, local scheduling strategy, and historical task execution statistics. This estimated computation start up time on resource i for a given task is called the *available time* of resource i for this task, denoted as σ_i.

There is a notable difference between the available time and the time when all jobs in the job queue complete execution when the resource is time-shared or space-shared. Multiple jobs run in parallel by multitasking or processors allocation making the resource available time for a new task much earlier than the time when all jobs in queue finish in many cases. The estimation is made by the local scheduler on the resource when receiving the task execution request from the task coordinator. The capability to estimate the available time of a resource allows us to reserve the resource predictably and achieve the improved makespan, since the resource with longer available time can be superior to the resource with immediate availability if the former shows much better computational capability.

2.3.1.4 Data Transfer Rate

data-intensive tasks require data to be replicated to geographically distributed resources segmentally or entirely. If data does not exist at the site where the task instance is supposed to be executed, the data need to be fetched from the data repository. This data staging process over the network connection will degrade the overall performance of the task execution and the cost on data staging contributes to the task makespan substantially if it cannot be eliminated or hidden from the makespan [79].

Several parameters such as bandwidth, latencies, packet loss, jitter and anomalies of network links affect data staging cost. The fourth performance evaluator, the *Data Transfer*

Rate R_i, characterizes the sustained data transfer rate between the resource i and the data repository for a given task. The data transfer rate is the comprehensive measurement on the network link performance involving all parameters mentioned above.

Several theoretical models predicting the bandwidth of a sustained TCP connection have been presented in the literature. In [80] the authors introduce a formula to derive the maximum TCP throughput subjected to light to moderate packet losses for TCP Congestion Avoidance algorithm [53, 101]. The end-to-end TCP/IP performance (bandwidth and latency) can also be returned by tools like *Network Weather Service (NWS)* [119] and *iperf* [52].

2.3.2 Application Model and Target Systems

We consider a data-intensive task $\mathcal{C}$ that consists of m independent duplicates, called m task instances (or process groups), all of which execute logically equivalent programs on different computing resources but each on a different input data segment: $\mathcal{C} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_m\}$. We assume that we have $n \geq m$ heterogeneous computing resources that form the *target systems set* $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2 \ldots \mathcal{S}_n\}$ which are able to execute the task $\mathcal{C}$. These resources can be owned by different persons, organizations, or institutions and thus are in different administrative domains but we assume that all of them can report the performance evaluators to the task coordinator.

A subset of target systems, forming the *restricted target set* $\mathcal{Q}^{(\pi)}$ such that $|\mathcal{Q}^{(\pi)}| = m$, will be selected by the scheduling algorithm from $\mathcal{S}$ to run the task. The task coordinator along with the m restricted target systems form the single level tree (star) topology as shown in Figure 2.2. There is a communication link from the task coordinator (the root of the tree) to each of the restricted target systems (the leafs of the tree), and the coordinator is able to dispatch the data in the data repository to the restricted target systems in parallel, resulting in a multi-ports communication model. Without loss of generality, we assume that

the coordinator has no data processing capability, thus all the workload will be processed by the m task instances mapping one to one onto the m restricted target systems. We name this scheduling model as *Divisible Load Scheduling (DLS)*.

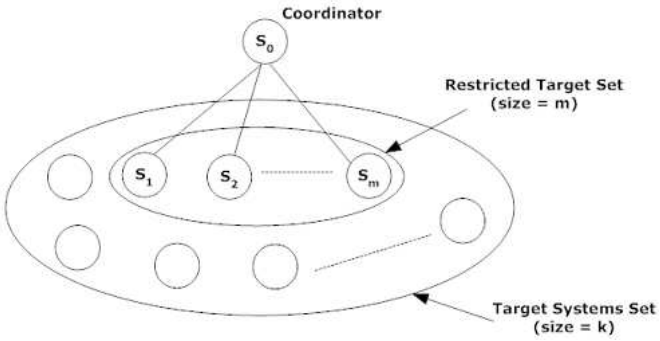


Figure 2.2: Single-level tree connection between task coordinator and m restricted target systems. The m restricted target system used for task parallel execution are selected from the target systems set.

Given the restricted target set $\mathcal{Q}^{(\pi)}$ with $|\mathcal{Q}^{(\pi)}| = m$, an *allocation* ν of process groups $\mathcal{G}_i \in \mathcal{C}$ to systems $\mathcal{S}_j \in \mathcal{Q}^{(\pi)}$ is an one-to-one mapping $\nu : \mathcal{G}_i \mapsto \mathcal{S}_j$. Given an allocation ν, the *data partitioning* for ν is to compute a decomposition δ of the input data set into segments of size $\omega_1, \omega_2, \ldots \omega_m$ such that $\omega = \sum_{i=1}^{m} \omega_i$ based on the objective function in such a way that process group $\mathcal{G}_i$ works on data segment ω_i. The pair $\pi = (\nu, \delta)$ describing both the allocation of process groups to target systems and the data partitioning is called a *mapping* of $\mathcal{C}$ to $\mathcal{Q}^{(\pi)}$.

For the sake of simplicity, we set to 0 the time when the task coordinator broadcasts the task execution request to the resources. In addition, we ignore the time spent on generating the schedule. This approximation is reasonable as the time spent on schedule generation can be ignored when comparing to the task execution time.

We define the *makespan* of a task execution as the elapsed time from the moment when the task execution request is sent out by coordinator to the resources to the moment when the task execution finishes, and denote it as $\tau^{(\pi)}$. Given a mapping π, the individual *process*

group completion time on the restricted target system $\mathcal{S}_i$ in $\mathcal{Q}^{(\pi)}$ is denoted as $T_i^{(\pi)}$. The objective function is defined as the makespan of task $\mathcal{C}$ under mapping π, denoted as:

$$\tau^{(\pi)} = \max(T_1^{(\pi)}, T_2^{(\pi)}, \ldots T_m^{(\pi)}). \quad \text{(Eq. 2.1)}$$

The DLS problem is to determine the optimal mapping such that the makespan $\tau^{(\pi)}$ of the task $\mathcal{C}$ is minimized:

$$\min_{\pi}[\tau^{(\pi)}]. \quad \text{(Eq. 2.2)}$$

If there are no restrictions regarding the maximum number of task instances/process groups, we can utilize as many target systems as we can in order to achieve the optimal makespan. The number of task instances/process groups becomes part of the output of the scheduling algorithm in this case. We call this scenario as *FlexMap*. In practice, we expect to be limited by factors such as the cost to access a resource, or some characteristics of the task that force us to use a definite number of target systems. The *FixMap* scenario addresses this problem and the given size of the process groups becomes the input of the scheduling algorithm instead.

2.3.3 Data Staging Strategies

Data staging affects the makespan of a task involving a large-scale input data set if it cannot be eliminated or hidden on the platform with low-speed and high-latency network connection. For systems utilizing the intelligent and adaptive resource management, it is not wise to replicate the entire input data set on each of the target systems in order to eliminate the data staging cost. The DLS meta-scheduler running on the coordinator will select a subset of target systems for the task execution after running the DLS algorithm and thus the approach of spreading the input data set to all target systems in advance may lead

to substantial storage resource waste and higher coordination overhead. Some suggest to schedule the data staging during the idle time [59, 43]; however, the available time and the data transfer rate are different for each system and this strategy requires special features of the execution engine on the resource that may, or may not, be available.

In this dissertation it is assumed that the data staging will not start until the coordinator finalizes the schedule/mapping, or in other words, before the coordinator receives acknowledgements of the resource reservation made from the peer systems in the restricted target set. Once a restricted target system has been selected, data staging can be initiated before its available time, after the available time, or after the available time but pipelining with the data processing, which leads to the different data staging strategies:

- Data Staging Before Available Time (DSBAT)

  As soon as the schedule/mapping is finalized by the coordinator, the data staging begins on the restricted target system. The data staging terminates on or before the available time and the computation starts immediately at the available time.

- Data Staging After Available Time (DSAAT)

  The data staging begins at the available time of the restricted target system. When data staging finishes, the computation begins immediately. This strategy corresponds to the one-round model in [12].

- Pipelined Data Staging (PDS)

  PDS model is an improved version of DSAAT model by overlapping the data staging with computing. The data staging is carried out by a p stage pipeline such that most of the data staging costs are hidden. This strategy corresponds to the multi-round model in [12].

Figure 2.3 shows the behavior of three data staging strategies. Intuitively DSBAT strategy can achieve the best makespan as it hides the data staging cost from the makespan

completely. However, this is not always the case and both DSAAT and PDS strategies have the opportunity to outperform the DSBAT strategy when the input data set size becomes large and the number of process groups is limited to a small number. This is due to the internal limitations on DSBAT strategy: (i) each restricted target system has the fixed data staging period which may limit the data a system can process even if it comes with the superior computational capability; (ii) the selection of the restricted target systems is also influenced by the fact that all the input data must be completely dispatched to them before their available times, otherwise, there is no solution for the DSBAT strategy. DSAAT strategy is the easiest scenario for implementation as it does not require the specific features of the local scheduler. The PDS strategy harnesses the pipeline execution by overlapping the data transfer and data processing under the condition that the data transfer rate R_i is larger or equal to the practical execution rate $\mu_i\eta_i$ for every restricted target system $\mathcal{S}_i$: $R_i >= \mu_i\eta_i$, such that the computation starts at each consecutive phase without waiting for the data. The DSAAT strategy is indeed a special case of the PDS strategy with the number of pipeline stages equal to 1 but without the condition on the data transfer rate and practical execution rate. We will further compare these three data staging strategies in detail in section 2.5.

Given the mapping π for a data-intensive task, suppose system $\mathcal{S}_i$ is selected as one restricted target system with the available time σ_i, the execution rate μ_i, the duty cycle η_i and the data transfer rate R_i. The data segment size assigned to $\mathcal{S}_i$ is denoted by $\omega_i^{(\pi)}$. Let δ_i and γ_i denote the data staging cost and data processing cost, respectively.

Under DSBAT strategy, the process group completion time $T_i^{(\pi)}$ will be:

$$T_i^{(\pi)} = \sigma_i + \gamma_i = \sigma_i + \frac{\omega_i^{(\pi)}}{\mu_i\eta_i} \qquad \text{(Eq. 2.3)}$$

with the assumption that $\frac{\omega_i^{(\pi)}}{R_i} <= \sigma_i$. This assumption guarantees that $\mathcal{S}_i$ finishes the data staging on or before its available time σ_i.

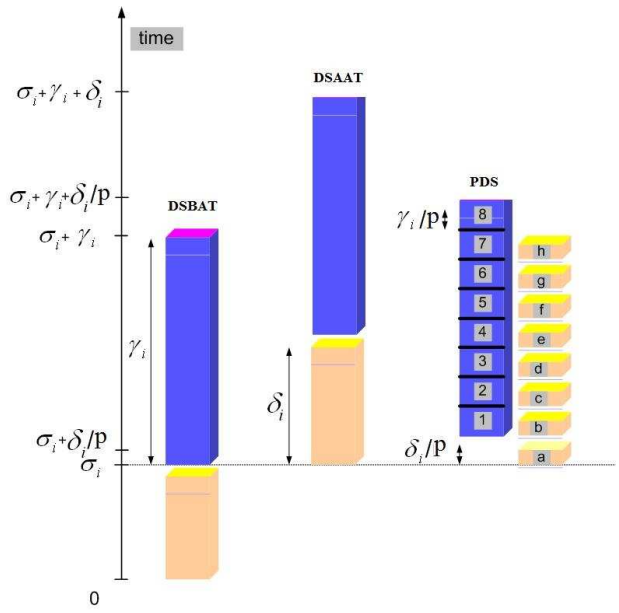


Figure 2.3: Computation of the individual process group completion time computed based on different data staging strategies on target system $\mathcal{S}_i$ with the available time σ_i: (Left) *DSBAT* Strategy - the data staging happens from the time when the mapping is generated to the available time of the target system. The process group completion time is $\sigma_i + \gamma_i$ with γ_i the data processing time. (Middle) *DSAAT* Strategy - The data staging starts at time σ_i and lasts δ_i seconds. The process group completion time is: $\sigma_i + \delta_i + \gamma_i$. (Right) *PDS* Strategy - The data staging occurs over p pipelined stages. The process group completion time is $\sigma_i + \delta_i/p + \gamma_i$. We let $p = 8$. Computation stages are labeled $1 - 8$ and the corresponding data staging are $a - h$.

Similarly, for the DSAAT strategy we have the process group completion time as:

$$T_i^{(\pi)} = \sigma_i + \delta_i + \gamma_i = \sigma_i + \frac{\omega_i^{(\pi)}}{R_i} + \frac{\omega_i^{(\pi)}}{\mu_i \eta_i}. \qquad \text{(Eq. 2.4)}$$

When the pipeline is applied, the process group completion time for the PDS strategy becomes:

$$T_i^{(\pi)} = \sigma_i + \delta_i + \gamma_i = \sigma_i + \frac{\omega_i^{(\pi)}}{pR_i} + \frac{\omega_i^{(\pi)}}{\mu_i \eta_i} \qquad \text{(Eq. 2.5)}$$

with p the number of pipeline stages.

Table 2.1: Divisible load scheduling (DLS) algorithms. σ_i is the available time of the restricted target system. DSBAT - data staging before available time; DSAAT - data staging after available time; PDS- pipelined data staging. FLX - FlexMap, FIX - FixMap.

Data Staging (DS)	No DS	DS prior σ_i	DS after σ_i	Pipelined DS
FlexMap	*FLX-NODS*	*FLX-DSBAT*	*FLX-DSAAT*	*FLX-PDS*
FixMap	*FIX-NODS*	*FIX-DSBAT*	*FIX-DSAAT*	*FIX-PDS*

2.4 Optimal Divisible Load Scheduling Algorithms

The limitation on the number of process groups or the target systems that the task can utilize combined with different data staging strategies leads to multiple families of algorithms. We summarize the families of algorithms in Table 2.1.

The following input parameters are required for the DLS algorithms.

- Target systems set $\mathcal{S}$ with performance evaluators μ_i, η_i, σ_i, R_i for each target system $\mathcal{S}_i$;
- Size of input data set ω;
- Size of restricted target set (*FixMap* scenario only) m;
- Number of pipeline stages (PDS model only) p.

The outputs of the DLS algorithms are:

- Task makespan $\tau^{(\pi)}$;
- Restricted target system set $\mathcal{S}^{(\pi)}$;
- Data segment size for each restricted target system $\omega_i^{(\pi)}$.

The intended objective within the scheduling process is to optimize the makespan $\tau^{(\pi)}$ of the data-intensive task. All DLS algorithms we present in this dissertation return the optimal schedule if the schedule exists under the given inputs. By optimal, we mean the schedule with the shortest makespan.

2.4.1 Divisible Load Scheduling Algorithms for DSBAT Data Staging Strategy

In this section, we discuss the Divisible Load Scheduling algorithms for the data staging before the available time (DSBAT) strategy. In addition, the *FLX-NODS* algorithm appears implicitly as a function in the description of the *FLX-DSBAT* algorithm and the *FIX-NODS* algorithm is introduced through a comparison with the *FIX-DSBAT* algorithm.

Although the data staging happens after the schedule is generated by the coordinator and before the available time on each target system, thus the data staging cost being hidden from the task makespan completely, the DSBAT scenario is significantly different from the case without data staging process for the following limitations: (i) the workload allocated to each restricted target system must be no more than the amount of input data it can receive during its data staging period; the maximal amount of data the restricted target system $\mathcal{S}_i$ can receive is calculated as $R_i\sigma_i$; (ii) the sum of the workload received by all restricted target systems needs to be equal to the input data set size, otherwise, there is no solution for the DSBAT scenario.

2.4.1.1 FLX-DSBAT Algorithm

The *FLX-DSBAT* scenario does not limit the number of the process groups and the number of target systems we can harness for task parallel execution. An iterative algorithm will be discussed but we first present the necessary and sufficient conditions for the optimal solution.

Proposition 1. Let $\mathcal{Q}^{(\pi)}$ be the restricted target set and $\tau^{(\pi)}$ be the task makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of *FLX-DSBAT* scenario if and only if three conditions are satisfied:

(i) *For each process group on the restricted target system in $\mathcal{Q}^{(\pi)}$, it finishes data staging on or before its available time:*

$$\sigma_i \geq \frac{\omega_i^{(\pi)}}{R_i}, \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}. \tag{Eq. 2.6}$$

(ii) *The process group completion time on each restricted target system in* $\mathcal{Q}^{(\pi)}$ *whose data staging process can finish earlier than its available time is equal to* $\tau^{(\pi)}$*:*

$$T_i^{(\pi)} = \tau^{(\pi)} \ , \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)} \ \ \textit{where} \ \ \sigma_i > \frac{\omega_i^{(\pi)}}{R_i}. \tag{Eq. 2.7}$$

(iii) *Any system* $\mathcal{S}_q$ *not in* $\mathcal{Q}^{(\pi)}$ *only becomes available after or at the task makespan under mapping* π*:*

$$\tau^{(\pi)} \leq \sigma_q, \quad \forall \mathcal{S}_q \notin \mathcal{Q}^{(\pi)}. \tag{Eq. 2.8}$$

Proof: We first show that the three conditions are necessary. Let $\pi = (\nu, \delta)$ be the optimal mapping for *FLX-DSBAT* scenario.

(i) Note that $\frac{\omega_i^{(\pi)}}{R_i}$ gives the time spent on the data staging. It is easy to see that Condition 1 holds based on the definition of *FLX-DSBAT* scenario.

(ii) Assume there exist two target systems $\mathcal{S}_x, \mathcal{S}_y \in \mathcal{Q}^{(\pi)}$ such that:

$$\sigma_x > \frac{\omega_x^{(\pi)}}{R_x}, \ \ \sigma_y > \frac{\omega_y^{(\pi)}}{R_y} \ , \textit{and} \ \ T_x^{(\pi)} < T_y^{(\pi)}.$$

Assume the process group completion time on target system $\mathcal{S}_z \in \mathcal{Q}^{(\pi)}$ is $\tau^{(\pi)}$. Note that $\mathcal{S}_z$ could be $\mathcal{S}_y$ or another target system, but in either case we have $T_x^{(\pi)} < T_z^{(\pi)} = \tau^{(\pi)}$. If we reassign a fraction of the workload from system $\mathcal{S}_z$ to $\mathcal{S}_x$, we end up achieving a new data partition scheme δ' corresponding to a new mapping $\pi' = (\nu, \delta')$ such that $T_x^{(\pi')} = T_z^{(\pi')} < \tau^{(\pi)}$.

This procedure can be repeated and eventually leads to a new task makespan $\tau^{(\pi')}$ with $\tau^{(\pi')} < \tau^{(\pi)}$ which contradicts the fact that π is an optimal mapping.

(iii) Assume there exists $\mathcal{S}_x \notin \mathcal{Q}^{(\pi)}$ such that $\sigma_x < \tau^{(\pi)}$. Now we can construct a new allocation ν' with $\mathcal{Q}^{(\pi')} = \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_x\}$. A new data partitioning δ' over $\mathcal{Q}^{(\pi')}$ will ensure a shorter makespan because some workload will be distributed to the new system. Thus the

new mapping $\pi' = (\nu', \delta')$ leads to a shorter makespan $\tau^{(\pi')} < \tau^{(\pi)}$ which contradicts the fact that π is an optimal mapping.

Next, we show that the three conditions are sufficient. Consider mapping π and assume that we have $\tau^{(\pi)}$ and $\mathcal{Q}^{(\pi)}$.

(i) As the condition $\sigma_i \geq \frac{\omega_i^{(\pi)}}{R_i}$, $\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}$ holds, all target systems in $\mathcal{Q}^{(\pi)}$ will finish data staging before or on their available times.

(ii) Assume all systems in $\mathcal{Q}^{(\pi)}$ satisfying the condition $\sigma_i > \frac{\omega_i^{(\pi)}}{R_i}$ form another target systems set $\mathcal{S}^{(t)}$ and finish computing at the same time, say $\tau^{(\pi)}$. Consider a new data partitioning δ' among systems in $\mathcal{S}^{(t)}$ and the corresponding mapping $\pi' = (\nu, \delta')$ with the same restricted target set. There will be at least one system, say $\mathcal{S}_x$ in $\mathcal{S}^{(t)}$, such that $\tau^{(\pi)} < T_x^{(\pi')} \leq \tau^{(\pi')}$, which means that redistributing data among systems in $\mathcal{S}^{(t)}$ cannot lead to a shorter computation completion time. As the systems in $\mathcal{Q}^{(\pi)} - \mathcal{S}^{(t)}$ cannot accept more data, the data redistribution between systems in $\mathcal{S}^{(t)}$ and systems in $\mathcal{Q}^{(\pi)} - \mathcal{S}^{(t)}$ will also lead to a longer makespan.

(iii) Consider the condition $\tau^{(\pi)} \leq \sigma_i, \ \forall \mathcal{S}_i \notin \mathcal{Q}^{(\pi)}$. It implies no new target system can be added into the the restricted target set $\mathcal{Q}^{(\pi)}$ to improve the makespan $\tau^{(\pi)}$. We conclude that π must be the optimal solution and finish the proof.

Algorithm 1 shows the pseudo-code of the *FLX-DSBAT* algorithm.

The first step of the *FLX-DSBAT* algorithm is to verify if a solution exists by accumulating the maximum data segment size that each target system in $\mathcal{S}$ can receive during its data staging process and comparing the result, expressed as:

$$\sum_{\forall \mathcal{S}_i \in \mathcal{S}} R_i \sigma_i, \qquad \text{(Eq. 2.9)}$$

with the input data set size ω. If it is less than ω, we are not able to process the whole input data set even if all target systems in $\mathcal{S}$ are utilized and thus there will be no solution for the given problem.

Input: Target systems set $\mathcal{S}$, task input data set size ω
Output: Task makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

if $\sum_{\forall \mathcal{S}_i \in \mathcal{S}} R_i\sigma_i \geq \omega$ **then**
  Initialize $\mathcal{S}^{(\pi)} \leftarrow \varnothing$;
  while *1* **do**
    Generate the temporary restricted target set $\mathcal{S}^{(p)}$ and temporary task makespan $\tau^{(p)}$ by running *FLX-NODS* algorithm on $\mathcal{S}$ and ω;
    Set flag = false;
    foreach *Target system* $\mathcal{S}_i$ *in temporary restricted target set* $\mathcal{S}^{(p)}$ **do**
      Calculate the data segment size allocated to $\mathcal{S}_i$ by *FLX-NODS* algorithm: ω_i=$\mu_i\eta_i(\tau^{(p)} - \sigma_i)$;
      Calculate the maximum data segment size $\mathcal{S}_i$ can get before its available time: d_i=$R_i\sigma_i$;
      if $\omega_i > d_i$ **then**
        Remove $\mathcal{S}_i$ from target systems set: $\mathcal{S} \leftarrow \mathcal{S} - \{\mathcal{S}_i\}$;
        Put $\mathcal{S}_i$ into final restricted target set: $\mathcal{S}^{(\pi)} \leftarrow \mathcal{S}^{(\pi)} \cup \{\mathcal{S}_i\}$;
        Reduce input data set size: $\omega = \omega - d_i$;
        Set flag = true;
      end
    end
    if *(flag == false)* **then**
      Set final restricted target set $\mathcal{S}^{(\pi)} \leftarrow \mathcal{S}^{(\pi)} \cup \mathcal{S}^{(p)}$;
      Set task makespan $\tau^{(\pi)} = \tau^{(\pi)}$;
      Exit while loop;
    end
  end
  foreach *target system* $\mathcal{S}_i$ *in* $\mathcal{S}^{(\pi)}$ **do**
    Calculate data segment size allocated to $\mathcal{S}_i$: $\omega_i^{(\pi)} = \min\left(R_i\sigma_i,\ \mu_i\eta_i(\tau^{(\pi)} - \sigma_i)\right)$;
  end
else
  No solution;
end

Algorithm 1: Pseudo-code for the *FLX-DSBAT* algorithm.

Assume the solution exists. The *FLX-DSBAT* algorithm will generate the temporary restricted target set $\mathcal{S}^{(p)}$ and temporary task makespan $\tau^{(p)}$ first by running the *FLX-NODS* algorithm without considering the data staging. The Algorithm 2 shows the pseudo-code of the *FLX-NODS* algorithm. The necessary and sufficient conditions for the optimal solution of *FLX-NODS* scenario is similar to Proposition 1 except that there is no limitation on the

size of the data segment each target system can get. The *FLX-NODS* algorithm iteratively adds new target systems which are ordered by their available times into the restricted target set and updates the task makespan accordingly until no more target systems can be involved into the restricted target set such that the makespan could be further improved. One key step of the *FLX-NODS* algorithm is to compute the task makespan after adding a new target system into the restricted target set. As the sum of the size of data segments allocated to restricted target systems should be equal to the task input data set size, we have the relation to compute the task makespan $\tau^{(j)}$ after the *jth* iteration of *FLX-NODS* algorithm:

$$\omega = \sum_{\forall \mathcal{S}_i \in \mathcal{S}^{(\pi)}} \mu_i \eta_i (\tau^{(j)} - \sigma_i), \qquad \text{(Eq. 2.10)}$$

with the task makespan $\tau^{(j)}$ the only unknown.

Input: Target systems set $\mathcal{S}$, task input data set size ω
Output: Task makespan $\tau^{(\pi)}$, restricted target set $\mathcal{S}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

Sort target systems in $\mathcal{S}$ based on their available times, the earliest first;
$\mathcal{S}^{(\pi)} \leftarrow \varnothing$;
for $i = 1$ *to SizeOf($\mathcal{S}$)* **do**
  Add system $\mathcal{S}_i$ into $\mathcal{S}^{(\pi)}$;
  Generate new task makespan $\tau^{(i)}$;
  if *Target system $\mathcal{S}_{i+1}$'s available time* $\geq \tau^{(i)}$ **then**
    Exit for loop;
  end
end
$\tau^{(\pi)} = \tau^{(i)}$;
foreach *Target system $\mathcal{S}_i$ in $\mathcal{S}^{(\pi)}$* **do**
  Compute data segment size: $\omega_i^{(\pi)} = \mu_i \eta_i (\tau^{(\pi)} - \sigma_i)$;
end

Algorithm 2: Pseudo-code for the *FLX-NODS* algorithm.

With the temporary restricted target set $\mathcal{S}^{(p)}$ and the temporary task makespan $\tau^{(p)}$, the corresponding data segment size allocated to the restricted target system $\mathcal{S}_i$, which is expressed as $\mu_i \eta_i (\tau^{(p)} - \sigma_i)$, is compared with the maximum data segment size $R_i \sigma_i$ it can

receive during its data staging period. If the former is larger, which indicates the coordinator allocates more data than $\mathcal{S}_i$ can actually receive before its available time, a *data set reduction* operation occurs by reducing $\mathcal{S}_i$'s workload to $R_i\sigma_i$, the maximum size of input data it can get. This data set reduction allows the target system $\mathcal{S}_i$ to begin computing at its available time σ_i. The necessity of data set reduction is checked at each iteration of the *FLX-DSBAT* algorithm and ensures that the first condition of Proposition 1 holds.

A consequence of reducing the data segment size assigned to $\mathcal{S}_i$ is the longer task makespan τ' because part of the workload on $\mathcal{S}_i$ needs to be transferred to the other target systems. Moreover, because of the data set reduction for $\mathcal{S}_i$, we have the relation $T_i < \tau^{(p)} < \tau'$ where T_i is the process group completion time for system $\mathcal{S}_i$ on the reduced data segment size $R_i\sigma_i$. This gives us the hint that $\mathcal{S}_i$ should be in the final restricted target set and the process group completion times of the systems in the restricted target set may not be equal anymore. Based on this observation, the restricted target systems experiencing the data set reduction are removed from the target systems set $\mathcal{S}$ and put into restricted target set $\mathcal{S}^{(\pi)}$. The workload they bear are deducted from the input data set ω as well. Then the algorithm continues to the next iteration and generates the new temporary restricted target set and temporary task makespan by *FLX-NODS* algorithm on the updated $\mathcal{S}$ and ω.

The algorithm continues until no data set reduction operations happen (the variable *flag* is used in the algorithm to identify this condition) on the target systems in the temporary target set $\mathcal{S}^{(p)}$, which indicates that all the systems in the temporary restricted target set from the output of *FLX-NODS* algorithm are able to finish their data staging before their individual available times; thus, by combining the $\mathcal{S}^{(p)}$ with the current $\mathcal{S}^{(\pi)}$ we achieve the final $\mathcal{S}^{(\pi)}$. This termination condition of *FLX-DSBAT* algorithm together with the properties of *FLX-NODS* algorithm guarantee that the conditions 2 and 3 of Proposition 1 hold.

The last step of the algorithm is to compute the data segment size allocated to each restricted target system. If the restricted target system $\mathcal{S}_i$ ever experiences the data set reduction, the workload on it will be the maximum input data size it can receive before its available time; otherwise, the workload is calculated by the simple derivation of the formula Eq. 2.3. In summary, we have the data segment size for $\mathcal{S}_i$ as

$$\omega_i^{(\pi)} = \min\left(R_i\sigma_i, \quad \mu_i\eta_i(\tau^{(\pi)} - \sigma_i)\right). \qquad \text{(Eq. 2.11)}$$

2.4.1.2 FIX-DSBAT Algorithm

The algorithms are more intricate when we limit the number of process groups, or the size of the restricted target set. Occasionally, the algorithm has to replace a system that is already included in the restricted target set by a better performing one to obtain a shorter makespan while keeping the size of restricted target set fixed. The available time of the target system becomes less important as in the *FlexMap* scenario since the system with longer available time can be superior to the one with much earlier availability if the former shows much better computational capability. To compensate for this restriction the algorithms for the *FixMap* scenario select the most "powerful" target systems, instead of using the systems with the earliest availability.

We now present the necessary and sufficient conditions for the optimal solution of the *FIX-DSBAT* scenario.

Proposition 2. Let $\mathcal{Q}^{(\pi)}$ be the restricted target set with fixed size m and $\tau^{(\pi)}$ be the makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of *FIX-DSBAT* scenario if and only if the following conditions are satisfied:

(i) *All process groups on the restricted target system in $\mathcal{Q}^{(\pi)}$, finishes the data staging on or before their available times:*

$$\sigma_i \geq \frac{\omega_i^{(\pi)}}{R_i} \ , \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}. \tag{Eq. 2.12}$$

(ii) *The process group completion time on each restricted target system in $\mathcal{Q}^{(\pi)}$ whose data staging process can finish earlier than its available time is equal to $\tau^{(\pi)}$:*

$$T_i^{(\pi)} = \tau^{(\pi)} \ , \quad \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)} \ \ \textit{where} \ \ \sigma_i > \frac{\omega_i^{(\pi)}}{R_i}. \tag{Eq. 2.13}$$

(iii) *Each system $\mathcal{S}_q$ outside the restricted target set under mapping π, either becomes available after or at the task makespan $\tau^{(\pi)}$:*

$$\tau^{(\pi)} \leq \sigma_q \tag{Eq. 2.14}$$

or, if $\tau^{(\pi)} > \sigma_q$, the following relation holds:

$$\min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \omega_i^{(\pi)} \geq \min\left((\tau^{(\pi)} - \sigma_q)\mu_q\eta_q, \ R_q\sigma_q\right). \tag{Eq. 2.15}$$

Proof: We first show that the three conditions are necessary. Let $\pi = (\nu, \delta)$ be an optimal mapping. Note that the proofs for necessity of Conditions 1 and 2 are exactly the same as their counterparts in the proof of Proposition 1. We discuss only the proof for the third condition.

Assume there is one $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ such that:

$$\tau^{(\pi)} > \sigma_q \ \ \textit{and} \ \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \omega_i^{(\pi)} < \min\left((\tau^{(\pi)} - \sigma_q)\mu_q\eta_q, \ R_q\sigma_q\right).$$

The expression $(\tau^{(\pi)} - \sigma_q)\mu_q\eta_q$ gives the projected data segment size that can be processed by $\mathcal{S}_q$ in the time interval $[\sigma_q, \tau^{(\pi)}]$ and the expression $R_q\sigma_q$ expresses the maximum size of data that can be transferred to $\mathcal{S}_q$ before its available time. The smaller one of these two values represents the actual size of data segment that $\mathcal{S}_q$ can process in the time interval $[\sigma_q, \tau^{(\pi)}]$ if DSBAT strategy is considered.

Given the mapping π with the task makespan $\tau^{(\pi)}$, assume that $\mathcal{S}_p \in \mathcal{Q}^{(\pi)}$ processes the least amount of data among systems in $\mathcal{Q}^{(\pi)}$. Thus, we have:

$$\omega_p^{(\pi)} = \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} \omega_i^{(\pi)} < \min\left((\tau^{(\pi)} - \sigma_q)\mu_q\eta_q,\ R_q\sigma_q\right), \qquad \text{(Eq. 2.16)}$$

which implies that when we use a new mapping π', generated by replacing $\mathcal{S}_p$ by $\mathcal{S}_q$, to process the same amount of input data set (equal to ω), the new task makespan $\tau^{(\pi')}$ will be less than $\tau^{(\pi)}$ by redistributing the input data among systems in $\mathcal{Q}^{(\pi')}$. This contradicts with the statement that $\tau^{(\pi)}$ is the optimal mapping.

Next, we show the sufficiency of these three conditions. Consider a mapping π and assume that we know $\tau^{(\pi)}$ and $\mathcal{Q}^{(\pi)}$. Condition 1 indicates that all systems in the restricted target set $\mathcal{Q}^{(\pi)}$ are able to finish their data staging on time and Condition 2 tells us that redistributing the data among systems in $\mathcal{Q}^{(\pi)}$ cannot lead to a shorter task makespan (see the proof of Proposition 1). Consider a system not in $\mathcal{Q}^{(\pi)}$. According to Condition 3, either the system becomes available after or at $\tau^{(\pi)}$ which means that adding it to $\mathcal{Q}^{(\pi)}$ cannot improve the makespan, or it is available before $\tau^{(\pi)}$, but the amount of data it can process is less than the amount any system in $\mathcal{Q}^{(\pi)}$ can process; thus it makes no sense to swap it with any system already in $\mathcal{Q}^{(\pi)}$. We conclude that π must be an optimal mapping. That finishes the proof.

Figure 2.4 provides an intuitive justification for the third condition of the Proposition 2. Consider five target systems $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_5\}$ ordered by their available times, the earliest first. Assume that the restricted target set is $\mathcal{Q}^{(\pi)} = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$ and that the *speed factor*

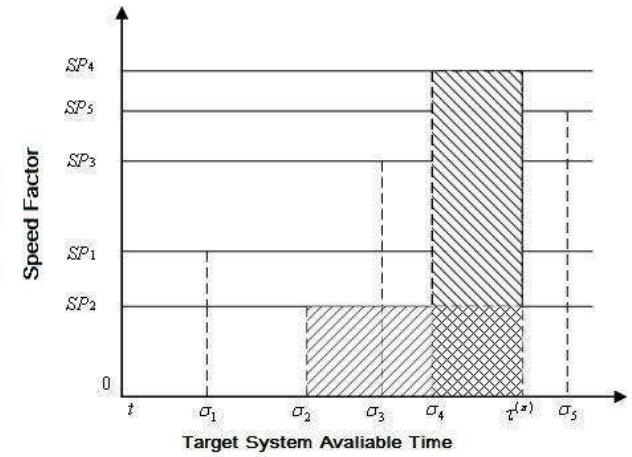


Figure 2.4: Comparison of systems workloads. The X-axis represents the available time σ_i, and the Y-axis the speed factor SP_i, equal to the average execution rate from σ_i to $\tau^{(\pi)}$, of five target systems $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_5\}$. The rectangular area bounded by $\tau^{(\pi)}$, σ_i and SP_i corresponds to $(\tau^{(\pi)} - \sigma_i)SP_i$, the workload allocated to and processed by $\mathcal{S}_i$. In this example the system $\mathcal{S}_4$ is able to accommodate a larger workload than $\mathcal{S}_2$; indeed, the area of the rectangle with length $(\tau^{(\pi)} - \sigma_4)$ and height SP_4 is larger than the area of the rectangle with length $(\tau^{(\pi)} - \sigma_2)$ and height SP_2.

SP_i, defined as $\mu_i\eta_i$ for DSBAT strategy, for the same Computation $\mathcal{C}$ on each system satisfy the condition: $SP_2 < SP_1 < SP_3 < SP_5 < SP_4$.

On the X-axis in Figure 2.4 we represent the available time $\sigma_i, 1 \leq i \leq 5$, of each system, and on the Y-axis the speed factor. If $\tau^{(\pi)}$ is the task makespan, the workload processed by the system $\mathcal{S}_i \in \mathcal{Q}^{(\pi)}$ is $(\tau^{(\pi)} - \sigma_i)SP_i$ according to the formula Eq. 2.3.

We observe that the rectangular area bounded by $\tau^{(\pi)}$, σ_i and SP_i in Figure 2.4 corresponds precisely to the workload allocated to and processed by the system $\mathcal{S}_i$. For example, the workload processed by $\mathcal{S}_2$ corresponds to the rectangle with the length $(\tau^{(\pi)} - \sigma_2)$ and height SP_2. The system $\mathcal{S}_5$ with the available time $\sigma_5 > \tau^{(\pi)}$ cannot be involved into the restricted target set. According to Figure 2.4, $\mathcal{S}_4$ can accommodate a larger workload than $\mathcal{S}_2$ because the area of the rectangle with length $(\tau^{(\pi)} - \sigma_4)$ and height SP_4 is larger than the area of the rectangle with length $(\tau^{(\pi)} - \sigma_2)$ and height SP_2. Thus replacing $\mathcal{S}_2$ with $\mathcal{S}_4$ in restricted target set will lead to a shorter task makespan.

The *FIX-DSBAT* algorithm involves three steps: (i) sort the target systems based on the maximum data segment size each system can receive during its data staging process and verify the existence of the solution for the given setup; (ii) construct an initial restricted target set $\mathcal{Q}^{(\pi)}$ with size m and refine the initial restricted target set by replacing systems in $\mathcal{Q}^{(\pi)}$ with systems outside $\mathcal{Q}^{(\pi)}$, such that the task makespan gets improved; and (iii) compute the data segment size for each system included in the final version of $\mathcal{Q}^{(\pi)}$. We show the pseudo-code of the *FIX-DSBAT* algorithm in Algorithm 3.

Input: Target systems set $\mathcal{S}$, task input data set size ω, the number of process groups m

Output: Task makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

Sort target systems in $\mathcal{S}$ based on the maximum data segment size each system can receive during its data staging process, the largest first;
if $\sum_{i=1}^{m} R_i\sigma_i \geq \omega$ **then**
    Put the first m target systems in the ordered $\mathcal{S}$ into restricted target set $\mathcal{Q}^{(\pi)}$;
    Compute task makespan $\tau^{(\pi)}$ by running *FLX-DSBAT* algorithm on $\mathcal{Q}^{(\pi)}$ and ω;
    while *1* **do**
        Generate candidates set $\mathcal{S}^{(c)}$;
        if $\mathcal{S}^{(c)} \neq \varnothing$ **then**
            Find system $\mathcal{S}_p$ in $\mathcal{Q}^{(\pi)}$ whose available time is before $\tau^{(\pi)}$ and processed the least workload among all systems in $\mathcal{Q}^{(\pi)}$;
            Find system $\mathcal{S}_q$ in $\mathcal{S}^{(c)}$ who will process the largest workload between time interval $[\sigma_q, \tau^{(\pi)}]$;
            if $\mathcal{S}_q$ *will process more data than* $\mathcal{S}_p$ **then**
                Remove $\mathcal{S}_p$ from $\mathcal{Q}^{(\pi)}$: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} - \{\mathcal{S}_p\}$;
                Put $\mathcal{S}_q$ into $\mathcal{S}^{(\pi)}$: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_q\}$;
                Generate new task makespan $\tau^{(\pi)}$ by running *FLX-DSBAT* algorithm on the updated $\mathcal{Q}^{(\pi)}$ and original ω;
            else
                Exit while loop;
            end
        else
            Exit while loop;
        end
    end
    foreach *Target system in restricted target set* $\mathcal{Q}^{(\pi)}$ **do**
        if *its available time is before task makespan* $\tau^{(\pi)}$ **then**
            Compute data segment size using formula Eq. 2.11;
        else
            Assign 0 workload to it;
        end
    end
else
    No solution;
end

Algorithm 3: Pseudo-code for the *FIX-DSBAT* algorithm.

Step 1 (Sorting): we first sort the systems in $\mathcal{S}$ based on the maximum amount of data that can be transferred to each one of them during the corresponding data staging phase which

is computed by expression $R_i\sigma_i$ for each system $\mathcal{S}_i$. The solution exists if the total amount of data that can be transferred during the data staging phase to the top m systems in the sorted set exceeds the total amount of input data ω.

Step 2 (Scheduling): Assume the solution exists. The initial restricted target set $\mathcal{Q}^{(\pi)}$ is constructed by including the top m systems in the sorted set $\mathcal{S}$. Then the *FLX-DSBAT* algorithm runs on restricted target set $\mathcal{Q}^{(\pi)}$ and input data set size ω which will return the initial task makespan $\tau^{(\pi)}$. With the initial task makspan, we can generate the *Candidates Set* $\mathcal{S}^{(c)}$ by including such systems that are not in $\mathcal{Q}^{(\pi)}$ but their available times are earlier than $\tau^{(\pi)}$. We call such a target system in $\mathcal{S}^{(c)}$ a *candidate system* because it is possible to obtain a shorter makespan if we swap it with one system already in $\mathcal{S}^{(\pi)}$ as indicated in Figure 2.4. We search for the system in $\mathcal{S}^{(c)}$ that has the largest estimated workload and for the system in $\mathcal{S}^{(\pi)}$ with the least workload assigned during the time interval from its available time to the current $\tau^{(\pi)}$ and swap them if the former processes more data. Note that both the estimated workload processed by the candidate system and the actual workload assigned to the target system in $\mathcal{S}^{(\pi)}$ are computed by formula Eq. 2.11. After the swap operation, the *FLX-DSBAT* algorithm is invoked to compute the new task makespan using the updated $\mathcal{S}^{(\pi)}$ and the original ω as inputs. With the new task makespan, the candidates set is regenerated and the possibility of a new swap operation is evaluated. The procedure is repeated until (1) there is no possibility of swap operation which indicates that the m most "powerful" systems have been selected already; or (2) the candidates set returns empty.

Step 3 (Data Partitioning): The workload for the restricted target system whose available time is earlier than the task makespan is computed by formula Eq. 2.11. For the systems which become available on or after the task makespan, we set their workload to 0.

We note that when the data staging cost is not considered, we have the *FIX-NODS* scenario. We can obtain the *FIX-NODS* algorithm by making several simple modifications on the *FIX-DSBAT* algorithm: (1) order the target systems based on their available times,

the earliest first, instead of the maximum input data size they can receive before their available times; (2) there is no need to verify the existence of the solution as the solution will always be reached; (3) both the estimate workload for system in candidates set and the actual workload for system in the restricted target set are computed as $\mu_i\eta_i(\tau^{(\pi)}-\sigma_i)$ instead of applying the formula Eq. 2.11.

2.4.2 Divisible Load Scheduling Algorithms for DSAAT and PDS Data Staging Strategies

As mentioned earlier, DSAAT strategy is a special case of PDS strategy. Therefore, in this section we focus on the algorithms for PDS strategy. We assume there are p pipeline stages and we will get the DSAAT strategy when $p = 1$.

2.4.2.1 FLX-PDS Algorithm

The *FLX-PDS* algorithm shares the same idea with the *FLX-NODS* and *FLX-DSBAT* algorithms. The target systems set is ordered by the target systems' available times. A new target system with the earliest available time is added into the restricted target set iteratively. The makespan of the task is updated during each iteration until no improvement is achieved. We present the sufficient and necessary conditions for the optimal solution of *FLX-PDS* scenario below.

Proposition 3. Let $\mathcal{Q}^{(\pi)}$ be the restricted target set and $\tau^{(\pi)}$ be the task makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of *FLX-PDS* scenario if and only if two conditions are satisfied:

(i) *The process group completion time, $\forall \mathcal{G}_i \in \mathcal{C}$, under mapping π, is the same and equal to the task makespan:*

$$T_1^{(\pi)} = T_2^{(\pi)} = ... = T_m^{(\pi)} = \tau^{(\pi)}. \qquad \text{(Eq. 2.17)}$$

(ii) *Any system $S_q \notin \mathcal{Q}^{(\pi)}$ only becomes available after or at the task makespan $\tau^{(\pi)}$:*

$$\tau^{(\pi)} \leq \sigma_q, \quad \forall \mathcal{S}_q \notin \mathcal{Q}^{(\pi)}. \tag{Eq. 2.18}$$

We omit the proof here as it is quite similar to the proof of Proposition 1.

Algorithm 4 shows the pseudo-code of *FLX-PDS* algorithm.

Input: Target systems set $\mathcal{S}$, input data set size ω, pipeline stage p
Output: Task makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

```
Sort target systems in S based on their available times, the earliest first;
Q^(π) ← ∅;
for i = 1 to SizeOf(S) do
    Add system S_i into Q^(π);
    Generate new task makespan τ^(i);
    if Target system S_{i+1}'s available time ≥ τ^(i) then
        Exit for loop;
    end
end
τ^(π) = τ^(i);
foreach Target system in Q^(π) do
    Compute data segment size using the derivation on the formula Eq. 2.5:
    ω_i^(π) = (τ^(π) − σ_i) / (1/(pR_i) + 1/(μ_i η_i));
end
```

Algorithm 4: Pseudo-code for the FLX-PDS algorithm.

The *FLX-PDS* algorithm is structured according to three steps.

Step 1 (Sorting): The target systems in the target systems set S are sorted based on their available times.

Step 2 (Scheduling): The target system with the earliest available time is iteratively added into the restricted target set $\mathcal{Q}^{(\pi)}$, which is initialized as an empty set. During each iteration, the task makespan is updated. Let $\tau^{(i)}$ denotes the task makespan after the ith iteration. For the first iteration, only one system, $\mathcal{S}_1$, is in $\mathcal{Q}^{(\pi)}$. The task makespan $\tau^{(1)}$ is computed

by formula Eq. 2.5 as

$$\tau^{(1)} = \sigma_1 + \delta_1 + \gamma_1 = \sigma_1 + \frac{\omega}{pR_1} + \frac{\omega}{\mu_1\eta_1}.$$

After putting $\mathcal{S}_2$ into $\mathcal{Q}^{(\pi)}$, the coordinator needs to relocate some workload from system $\mathcal{S}_1$ to $\mathcal{S}_2$ and we denote the amount of workload reallocated to $\mathcal{S}_2$ from $\mathcal{S}_1$ as Δ_1. Note that the new task makespan $\tau^{(2)}$ can be expressed by formula Eq. 2.5 as

$$\tau^{(2)} = \sigma_2 + \delta_2 + \gamma_2 = \sigma_2 + \frac{\Delta_1}{pR_2} + \frac{\Delta_1}{\mu_2\eta_2}.$$

Considering the improvement of the task makespan, we have another way to express $\tau^{(2)}$ as

$$\tau^{(1)} - \tau^{(2)} = \frac{\Delta_1}{pR_1} + \frac{\Delta_1}{\mu_1\eta_1}.$$

From these two different expressions of $\tau^{(2)}$, we derive the following formulas

$$\Delta_1 = \frac{\tau^{(1)} - \sigma_2}{(\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}) + (\frac{1}{pR_2} + \frac{1}{\mu_2\eta_2})} \qquad \text{(Eq. 2.19)}$$

and

$$\tau^{(2)} = \tau^{(1)} - (\frac{\Delta_1}{pR_1} + \frac{\Delta_1}{\mu_1\eta_1}). \qquad \text{(Eq. 2.20)}$$

Similar to the derivation of $\tau^{(2)}$, assume we have j systems in the restricted target set, $\mathcal{S}_1, \mathcal{S}_2, ...\mathcal{S}_j$, where $j > 2$, and the task makespan is $\tau^{(j)}$. To update the task makespan after adding the system $\mathcal{S}_{j+1}$, we have the following observations:

(i) The process group completion time on each of the first j systems in the restricted target set is improved by the same amount:

$$\Delta_1 \left(\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}\right) = \Delta_2 \left(\frac{1}{pR_2} + \frac{1}{\mu_2\eta_2}\right) = ... = \Delta_j \left(\frac{1}{pR_j} + \frac{1}{\mu_j\eta_j}\right). \qquad \text{(Eq. 2.21)}$$

(ii) The total workload reallocated from the first j systems in the restricted target set to the newly added system $\mathcal{S}_{j+1}$ is:

$$\omega_{j+1} = \sum_{i=1}^{j} \Delta_i. \qquad \text{(Eq. 2.22)}$$

Note that from formula Eq. 2.5 we have another way to express the ω_{j+1} as

$$\omega_{j+1} = \frac{\tau^{(j+1)} - \sigma_{j+1}}{\frac{1}{pR_{j+1}} + \frac{1}{\mu_{j+1}\eta_{j+1}}}. \qquad \text{(Eq. 2.23)}$$

(iii) The task makespan is improved by the amount equal to the improvement of the process group completion time on each of the first j systems:

$$\tau^{(j)} - \tau^{(j+1)} = \Delta_i \left(\frac{1}{pR_i} + \frac{1}{\mu_i \eta_i} \right), \quad 1 \leq i \leq j. \qquad \text{(Eq. 2.24)}$$

The formulas Eq. 2.21, Eq. 2.22, Eq. 2.23, and Eq. 2.24 allow us to derive the expressions for Δ_1 and $\tau^{(j+1)}$ as:

$$\Delta_1 = \frac{\tau^{(j)} - \sigma_{j+1}}{(\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}) + (\frac{1}{pR_{j+1}} + \frac{1}{\mu_{j+1}\eta_{j+1}})(1 + \sum_{i=2}^{j}(\frac{\frac{1}{pR_1} + \frac{1}{\mu_1\eta_1}}{\frac{1}{pR_i} + \frac{1}{\mu_i\eta_i}}))} \qquad \text{(Eq. 2.25)}$$

and

$$\tau^{(j+1)} = \tau^{(j)} - (\frac{\Delta_1}{pR_1} + \frac{\Delta_1}{\mu_1\eta_1}). \qquad \text{(Eq. 2.26)}$$

The iteration terminates when there is no more improvement in the task makespan or no more target systems are available with the available times earlier than the current task makespan.

Step 3 (Data Partitioning): After achieving the task makespan $\tau^{(\pi)}$ from Step 2, the size of the data segment allocated to the restricted target system $\mathcal{S}_i$ can be computed by the

following formula derived from formula Eq. 2.5:

$$\omega_i^{(\pi)} = \frac{\tau^{(\pi)} - \sigma_i}{\frac{1}{pR_i} + \frac{1}{\mu_i \eta_i}}. \qquad \text{(Eq. 2.27)}$$

2.4.2.2 FIX-PDS Algorithm

The *FIX-PDS* algorithm constrains the number of process groups m, or in other words, the number of target systems that the coordinator can utilize for task parallel execution. As before, the optimality of mapping ensures the shortest makespan of the task, but in this case, we map the process groups to precisely m target systems.

The idea of the *FIX-PDS* algorithm is to generate an initial restricted target set $\mathcal{Q}^{(\pi)}$ of size m first by iteratively adding to $\mathcal{Q}^{(\pi)}$ new systems. Then, the *candidates set* $\mathcal{S}^{(c)}$ is constructed after we compute the initial $\mathcal{Q}^{(\pi)}$ and $\tau^{(\pi)}$. All target systems with available times earlier than the current $\tau^{(\pi)}$ will be added into $\mathcal{S}^{(c)}$. We call such a target system a *candidate* because it is possible to obtain a shorter task makespan if we swap it with one system already in $\mathcal{Q}^{(\pi)}$. We compare the system in $\mathcal{S}^{(c)}$ that has the largest estimated workload with the system in $\mathcal{Q}^{(\pi)}$ with the least workload assigned during the time interval from its available time to the current $\tau^{(\pi)}$ and swap them if the former has a larger workload. After each swap operation, the task makespan is updated and the candidates set is regenerated. The iterative process ends when no such swap operation is possible. The last step is to compute the workload allocated to each restricted target system in $\mathcal{Q}^{(\pi)}$.

The necessary and sufficient conditions for the optimality of the solution from the *FIX-PDS* scenario are summarized by the following proposition.

Proposition 4. Let $\mathcal{Q}^{(\pi)}$ be the restricted target set with fixed size m and $\tau^{(\pi)}$ be the task makespan under mapping π. The mapping $\pi : \mathcal{C} \mapsto \mathcal{Q}^{(\pi)}$ is the optimal solution of *FIX-PDS* scenario if and only if two conditions are satisfied:

(i) *The process group completion time* $T_i^{(\pi)}$, $\mathcal{G}_i \in \mathcal{C}$, *under mapping* π, *is the same and equal to the task makespan:*

$$T_1^{(\pi)} = T_2^{(\pi)} = ... = T_m^{(\pi)} = \tau^{(\pi)}. \quad \text{(Eq. 2.28)}$$

(ii) *Each system outside the restricted target set under mapping* π, $\forall \mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$, *either becomes available after or at the task makespan* $\tau^{(\pi)}$: $\tau^{(\pi)} \leq \sigma_q$, *or, if* $\tau^{(\pi)} > \sigma_q$, *the following relation holds:*

$$\min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} (\tau^{(\pi)} - \sigma_i) \frac{1}{(\frac{1}{pR_i} + \frac{1}{\mu_i \eta_i})} \geq (\tau^{(\pi)} - \sigma_q) \frac{1}{(\frac{1}{pR_q} + \frac{1}{\mu_q \eta_q})}. \quad \text{(Eq. 2.29)}$$

Proof. We first show that the two conditions are necessary. Let $\pi = (\nu, \delta)$ be the optimal mapping for the *FIX-PDS* scenario.

(i) Assume two process groups, $\mathcal{G}_x, \mathcal{G}_y \in \mathcal{C}$ exist such that, under the allocation ν corresponding to the mapping π with task makespan $\tau^{(\pi)}$, one process group, $\mathcal{G}_x$, is able to finish earlier than the other: $T_x^{(\pi)} < T_y^{(\pi)}$ and $T_y^{(\pi)} = \tau^{(\pi)}$. If we reassign a fraction of the workload from system $\mathcal{S}_y$ to $\mathcal{S}_x$, we may achieve the new data partition scheme δ' corresponding to a new mapping $\pi' = (\nu, \delta')$ such that: $T_x^{(\pi')} = T_y^{(\pi')} < \tau^{(\pi)}$. This process can be repeated until the completion time of each process group becomes the same and thus will lead to a new task makespan: $\tau^{(\pi')} < \tau^{(\pi)}$, which contradicts with the assumption that $\tau^{(\pi)}$ is optimal.

(ii) Assume $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ and satisfies the relation:

$$\tau^{(\pi)} > \sigma_q \quad and \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} (\tau^{(\pi)} - \sigma_i) \frac{1}{(\frac{1}{pR_i} + \frac{1}{\mu_i \eta_i})} < (\tau^{(\pi)} - \sigma_q) \frac{1}{(\frac{1}{pR_q} + \frac{1}{\mu_q \eta_q})}.$$

The expression $(\tau^{(\pi)} - \sigma_q) \frac{1}{(\frac{1}{pR_q} + \frac{1}{\mu_q \eta_q})}$ gives the workload that can be processed by $\mathcal{S}_q$ within the time interval $[\sigma_q, \tau^{(\pi)}]$, based on the formula Eq. 2.27.

Given the mapping π with the task makespan $\tau^{(\pi)}$, assume that $\mathcal{S}_p \in \mathcal{Q}^{(\pi)}$ processes the lowest workload among systems in $\mathcal{Q}^{(\pi)}$. Thus, we have:

$$(\tau^{(\pi)} - \sigma_p)\frac{1}{(\frac{1}{pR_p} + \frac{1}{\mu_p\eta_p})} = \min_{\forall \mathcal{S}_i \in \mathcal{Q}^{(\pi)}} (\tau^{(\pi)} - \sigma_i)\frac{1}{(\frac{1}{pR_i} + \frac{1}{\mu_i\eta_i})} < (\tau^{(\pi)} - \sigma_q)\frac{1}{(\frac{1}{pR_q} + \frac{1}{\mu_q\eta_q})}.$$

This implies that if we create a new mapping π', generated by replacing $\mathcal{S}_p$ by $\mathcal{S}_q$, for the same input workload ω, then new makespan $\tau^{(\pi')} < \tau^{(\pi)}$. This contradicts the statement that $\tau^{(\pi)}$ is optimal.

Next we show that the two conditions are sufficient.

(i) Assume all systems in $\mathcal{Q}^{(\pi)}$ finish computation at the same time $\tau^{(\pi)}$. Consider a new data partitioning δ' and the corresponding mapping $\pi' = (\nu, \delta')$ with the same restricted target set. There will be at least one system, say $\mathcal{S}_x$, such that: $\tau^{(\pi)} < T_x^{(\pi')} \leq \tau^{(\pi')}$, which will lead to a longer task makespan.

(ii) Consider a mapping π and a system $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ such that $\sigma_q \geq \tau^{(\pi)}$. Thus, it would not be helpful to add $\mathcal{S}_q$ to $\mathcal{Q}^{(\pi)}$. On the other hand, assume $\mathcal{S}_q \notin \mathcal{Q}^{(\pi)}$ has the available time $\sigma_q < \tau^{(\pi)}$. The second condition of the proposition tells us that the workload that can be processed by $\mathcal{S}_q$ from its available time to the current $\tau^{(\pi)}$ is less than or equal to the minimum workload allocated to any system in $\mathcal{Q}^{(\pi)}$. This implies that swapping $\mathcal{S}_q$ with any system in $\mathcal{Q}^{(\pi)}$ cannot shorten the task makespan. We finish the proof.

Algorithm 5 shows the pseudo-code of *FIX-PDS* algorithm.

The *FIX-PDS* algorithm involves four steps.

Step 1 (Sorting): The target systems in the target systems set S is sorted based on their available times, the earliest first.

Step 2 (Initializing $\mathcal{Q}^{(\pi)}$): We need to create the initial restricted target set $\mathcal{Q}^{(\pi)}$ with size m. The target system with the earliest available time is added into $\mathcal{Q}^{(\pi)}$ iteratively. During each iteration, the task makespan is updated. The algorithm for the task makespan update

is similar to the one introduced in *FLX-PDS* algorithm (formula Eq. 2.5, formulas Eq. 2.19 and Eq. 2.20, and formulas Eq. 2.25 and Eq. 2.26). It is possible that after k ($k < m$) iterations, the task makespan is already shorter than or equal to the available time of any target system not in the restricted target set, which indicates that no improvement will be achieved by adding more target systems into $\mathcal{Q}^{(\pi)}$. There are two options in dealing with this situation: (i) we simple add the next $m - k$ target systems into $\mathcal{Q}^{(\pi)}$ from the ordered set $\mathcal{S}$ such that the size m is reached. We call these systems as "dummy restricted target systems" as they become available after the task makespan thus no formal resource reservations will be made on them and no workload will be actually distributed to them; or (ii) we update m to the smaller value k and thus reduce the size of the restricted target set and the number of process groups. The *FIX-PDS* algorithm we present here implements the first strategy but it can be easily modified to support the second strategy by simply replacing the clause of adding $m - k$ system into $\mathcal{Q}^{(\pi)}$ by the clause of updating value of m to k.

Input: Target systems set $\mathcal{S}$, input data set size ω, the number of process groups m, pipeline stage p

Output: Task makespan $\tau^{(\pi)}$, restricted target set $\mathcal{Q}^{(\pi)}$, data segment size for each restricted target system $\omega_i^{(\pi)}$

Sort target systems in $\mathcal{S}$ based on their available times, the earliest first;
$\mathcal{Q}^{(\pi)} \leftarrow \varnothing$;
for $i = 1$ *to* m **do**
    Add system $\mathcal{S}_i$ into $\mathcal{Q}^{(\pi)}$;
    Generate new task makespan $\tau^{(\pi)}$;
    if *(Target system* $\mathcal{S}_{i+1}$*'s available time* $\geq \tau^{(\pi)}$*) and (*$i < m$*)* **then**
        Add systems $\mathcal{S}_{i+1}, \mathcal{S}_{i+2} ... \mathcal{S}_m$ into $\mathcal{Q}^{(\pi)}$;
        Exit for loop;
    end
end
while *1* **do**
    Generate candidates set $\mathcal{S}^{(c)}$;
    if $\mathcal{S}^{(c)} \neq \varnothing$ **then**
        Find system $\mathcal{S}_p$ in $\mathcal{Q}^{(\pi)}$ whose available time is before $\tau^{(\pi)}$ and processes the least workload among all systems in $\mathcal{Q}^{(\pi)}$;
        Find system $\mathcal{S}_q$ in $\mathcal{S}^{(c)}$ who can process the most workload from its available time σ_q to $\tau^{(\pi)}$;
        if $\mathcal{S}_q$ *will process more data than* $\mathcal{S}_p$ **then**
            Remove $\mathcal{S}_p$ from $\mathcal{Q}^{(\pi)}$: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} - \{\mathcal{S}_p\}$;
            Put $\mathcal{S}_q$ into $\mathcal{Q}^{(\pi)}$: $\mathcal{Q}^{(\pi)} \leftarrow \mathcal{Q}^{(\pi)} \cup \{\mathcal{S}_q\}$;
            Generate new task makespan $\tau^{(\pi)}$;
        else
            Exit while loop;
        end
    else
        Exit while loop;
    end
end
foreach *Target system in restricted target set* $\mathcal{Q}^{(\pi)}$ **do**
    if *Its available time is before the task makespan* $\tau^{(\pi)}$ **then**
        Compute data segment size using formula Eq. 2.27;
    else
        Assign zero workload to it;
    end
end

Algorithm 5: Pseudo-code for the *FIX-PDS* algorithm.

Step 3 (Scheduling): The candidates set $\mathcal{S}^{(c)}$ is constructed starting with the initial restricted target set and task makespan. All target systems not in $\mathcal{Q}^{(\pi)}$ but with available time earlier than current $\tau^{(\pi)}$ are put into $\mathcal{S}^{(c)}$. Then, we search for the system $\mathcal{S}_p$ in $\mathcal{Q}^{(\pi)}$ which processes the least workload and for the system $\mathcal{S}_q$ in $\mathcal{S}^{(c)}$ which will process the most workload from its available time to current $\tau^{(\pi)}$. The workload of the systems are either estimated or computed by applying the formula Eq. 2.27. If the estimated workload of $\mathcal{S}_q$ is larger than the actual workload processed by $\mathcal{S}_p$, we remove $\mathcal{S}_p$ from the restricted target set and put $\mathcal{S}_q$ in. As indicated in Figure 2.4 (speed factor SP_i becomes $\frac{1}{(\frac{1}{pR_i}+\frac{1}{\mu_i\eta_i})}$ in this case), swapping $\mathcal{S}_p$ and $\mathcal{S}_q$ leads to a better task makespan.

After the restricted target set $\mathcal{Q}^{(\pi)}$ is updated, the task makespan is recomputed by simulating the *FLX-PDS* scenario on $\mathcal{Q}^{(\pi)}$ with the following steps: (i) order the systems in $\mathcal{Q}^{(\pi)}$ based on their available times, the earliest first; (ii) iteratively add the system with the earliest available time into the "restricted target set" of $\mathcal{Q}^{(\pi)}$ and compute the task makespan based on formula Eq. 2.5 for the first iteration, formulas Eq. 2.19 and Eq. 2.20 for the second iteration, and formulas Eq. 2.25 and Eq. 2.26 for the other iterations.

With the updated task makespan, the algorithm reconstructs the candidates set and repeats the procedure in Step 3. The iteration terminates when (i) the candidates set becomes an empty set; or (ii) no swap operation is possible which indicates that the maximum estimated workload processed by any system in the candidates set is less than or equal to the minimum workload handled by any system in the restrict target set.

Step 4 (Data Partitioning): Computing the data partitions is the last step of the algorithm. It is possible that $\mathcal{Q}^{(\pi)}$ involves systems that will be available after or at the time $\tau^{(\pi)}$; we identify them and set the workload assigned to them to zero. For the other restricted target systems, we calculate the data segment size allocated to them based on formula Eq. 2.27.

2.4.3 Fault-tolerant Coordination Algorithms

The computations considered in this dissertation take a very long time and the probability that one of the target systems experiences a failure during this time cannot be neglected. Once a systems fails, the coordination algorithm redistributes the work originally allocated to the failing system to the systems already in the restricted target set, or attempts to reassign the work to other systems in the original target set.

```
Run the FlexMap/FixMap algorithm;
Spawn m process groups, G_1, G_2...G_m, one for each restricted target system identified
by the FlexMap/FixMap algorithm;
Set completion = false;
while completion == false do
    if Fault detected on G_i then
        Construct the new restricted target set by removing system where G_i resides;
        Determine the amount of data needed to be redistributed;
        Setup the startup times for systems in the new restricted target set to be
        current computation completion time;
        Run the FlexMap/FixMap algorithm to compute new data partitions;
    end
    if All process groups have signaled completion then
        Set completion = true;
    end
end
```

Algorithm 6: The fault-tolerance coordination algorithm on coordination site.

In this section we sketch the coordination algorithms that redistribute the work originally allocated to the failing system to the systems already in the restricted target set. The basic idea of the fault-tolerant coordination is to detect the failure by estimating the completion time of each pipeline stage on each system in the restricted target set and expect an acknowledgment that the stage has finished. We assume that the partial results for each stage are sent by each system in the restricted target set at the end of each stage to the coordinator together with the acknowledgment. If such an acknowledgment is missing then:

```
Compute the time per pipeline stage;
Wait for the startup time on the restricted target system S_i;
Send the first block of data;
Startup the computation on system S_i;
Set the timer for an interrupt at the end of first pipeline stage;
Send the second block of data;
for k=1 to the number of pipeline stages - 1 do
    Wait for partial results;
    if timer interrupt then
        Send to coordination program "fault at system S_i" signal;
        Kill process group G_i and exit;
    else
        Store partial results;
        Send the next block of data;
        Set a timer for an interrupt at the end of the next stage;
    end
end
Wait for partial results;
if timer interrupts then
    Send to coordination program "fault at system S_i" signal;
    Kill process group G_i;
else
    Store partial results;
    Send to coordination program "complete at system S_i" signal;
end
```

Algorithm 7: The fault-tolerance coordination algorithm to manage the execution of process group $\mathcal{G}_i$ on target system $\mathcal{S}_i$.

- If the restricted target set is $\mathcal{Q}^{(\pi)} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_p, \ldots \mathcal{S}_m\}$ and $\mathcal{S}_p$ is the failing system, we determine the amount of the data allocated initially to system $\mathcal{S}_p$ that is still unprocessed. Call this amount Δ_p.

- If the original computation completion time estimated by the algorithm was $\tau^{(\pi)}$ then we run the *FlexMap/FixMap* divisible load co-scheduling algorithm for the $(m-1)$ systems in the new restricted target set $\mathcal{Q}^{(\pi')} = \mathcal{Q}^{(\pi)} - \{\mathcal{S}_p\}$. The new startup time $\sigma_i' = \tau^{(\pi)}, \ \ \forall \mathcal{S}_i \in \mathcal{Q}^{(\pi')}$. The amount of input data is equal to Δ_p.

- The algorithm produces an additional amount of data to be sent to each system and a new computation completion time $\tau^{(\pi')} = \tau^{(\pi)} + \Delta_\tau$.

The pseudo-code for the coordination algorithms are shown in Algorithm 6 and Algorithm 7.

The fault-tolerant coordination algorithms we just present maintain the resource utilization rate (the number of target systems used for the computation). Another strategy to provide the fault-tolerance computation is to exploit the idea of duplicate computation. After the optimal mapping is generated, for each restricted target system we select a backup system with the similar parameters. Each backup system holds the same amount of workload as its corresponding restricted target system, and will start data processing at its available time. Thus, if a restricted target system fails, the coordinator can gather the computation result from the backup system instead of redistributing the unprocessed data to the other systems. This strategy limits potentially the increase of the task makespan, if the parameters of the backup system is close enough to the parameters of the corresponding restricted target system.

2.5 Performance Evaluation

In this section we first study the behavior of the *FlexMap* and *FixMap* algorithms by simulation and then compare our divisible load co-scheduling model under multi-port communication strategy with the divisible load model with one-port communication.

2.5.1 Simulation Study

We simulate an ensemble of 200 target systems and investigate the task makespan in function of the input data set size and the number of process groups m. We also observe the size of the restricted target set for the *FlexMap* scenario. Each target system is characterized by a vector consisting of the available time σ, the execution rate μ, the duty cycle η, and the data transfer rate R. These four random variables are normally distributed; the mean and the standard deviations of the four random variables are, respectively: 25 and 5 hours for the

available time σ, 450 and 80 Mdtu/hour for the execution rate μ, 0.75% and 0.05% for the duty cycle η, and, finally, 550 and 30 Mdtu/hour for the data transfer rate R. The average execution rate and the data transfer rate are related to the biological application discussed in [54, 129] and measured by running the test process on the multiple clusters in University of Central Florida. The available time and the duty cycle are selected based on the analysis of the trace of workload on each cluster.

In our simulation we first construct 200 random vectors $S_i(\mu_i, \sigma_i, \eta_i, R_i), \ \forall \mathcal{S}_i \in \mathcal{S}$ which characterize the target systems set $\mathcal{S}$. Then the DLS algorithms select the restricted target set $\mathcal{Q}^{(\pi)}$, compute task makespan and the amount of data allocated to each system. The DLS algorithms are deterministic, thus for a given target systems set configuration and input data set size, the task makespan and the generated restricted target set are the same for each run.

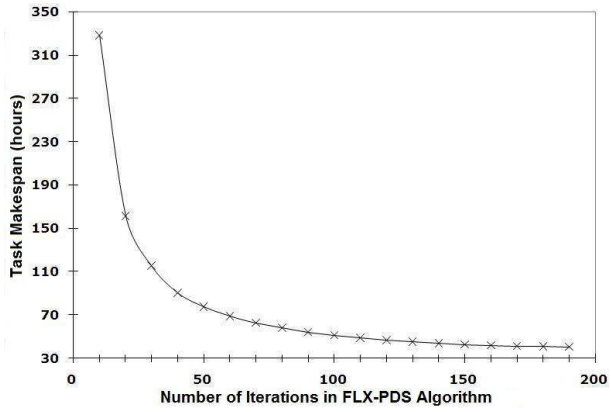


Figure 2.5: Task makespan (in hours) function of the number of iterations for a fixed input data set size, $\omega = 10^{12}$ *dtu*, for the *FLX-PDS* algorithm.

Figure 2.5 shows the task makespan function of the number of iterations of the first *for* loop for the *FLX-PDS* algorithm. At each iteration a new target system is added into the restricted target set if the available time of that system is earlier that the current task makespan; thus, the task makespan monotonously decreases as the iteration process progresses. Figure 2.6 (Left) and Figure 2.6 (Right) show the first 15 and the last 15

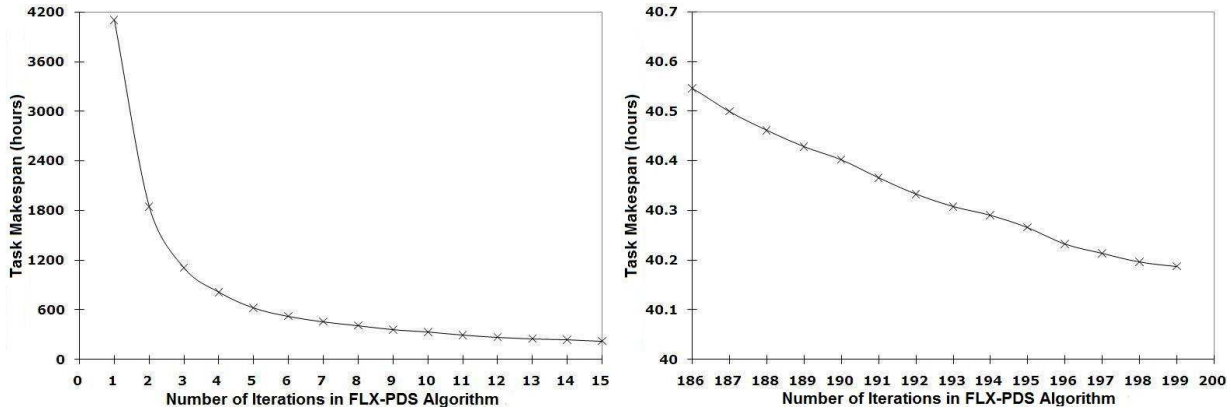


Figure 2.6: Task makespan (in hours) for the *FLX-PDS* algorithm. (Left) The first 15 iterations. (Right) The last 15 iterations.

iterations, respectively. If we define the speedup as the ratio of the task makespan at the first iteration of an interval to the one at the last iteration of the interval, we notice that:

$$Speedup_{1-15} = \frac{4103}{221} = 18.57$$

while

$$Speedup_{186-200} = \frac{40.58}{40.19} = 1.01.$$

This indicates that the benefits of using an increasingly larger number of systems have to be balanced against the additional cost and overhead to coordinate multiple sites. We conclude that the *FLX-PDS* algorithm should have an additional termination condition:

$$\text{if } (\tau^{(j)}/\tau^{(j-1)} < MinSpeedupPerIteration) \quad \text{terminate;}$$

We note that this conclusion applies to all *FlexMap* family of algorithms (*FLX-NODS*, *FLX-DSBAT*, *FLX-DSAAT*, and *FLX-PDS*) as they share the same idea in building up the optimal solution.

We now compare the three different data staging strategies for the *FlexMap* family of algorithms and Figure 2.7 shows the results. The three flavors of the algorithm end up using

all 200 target systems as soon as the input data set size reaches 11×10^{11} *dtu*, 19×10^{11} *dtu*, and 5×10^{11} *dtu*, respectively. The *FLX-DSAAT* leads to the longer task makespan (see Figure 2.7 (Left)) for each input data size and requires more resources. The *FLX-DSBAT* cannot guarantee the existence of the solution when the input data set size is relatively large (in our case when the input data set size reaches 21×10^{11} *dtu*) and the target systems set is relatively small [128], while the other two succeed. The *FLX-PDS* algorithm reduces the task makespan substantial comparing to *FLX-DSAAT* algorithm and places a lower load upon resources.

As expected, the number of systems used for a given task increases as the input data set size increases as shown in the Figure 2.7 (Right). In this experiment, the resource vectors $(\mu_i, \sigma_i, \eta_i, R_i),\ \forall \mathcal{S}_i \in \mathcal{S}$ are the same, only the data set size increases. For a relatively small input data set size, after a few iterations, the task makespan becomes shorter than the available times of the systems outside of the restricted target set. As the input data set size increases, the task makespan after the same number of iterations increases and allows us to include more systems in the restricted target set.

The more complex *FixMap* family of algorithms is discussed next. First, we study the the effect of the number p of pipeline stages. As expected, the larger the number p, the shorter is the time to fill-in the pipeline and start the computation and the shorter is the task makespan, Figure 2.8.

The effects of the data staging when we limit the number of systems included in the restricted target set are summarized in Figure 2.9 for the three flavors of the algorithm. The pipelined version is better than *FIX-DSBAT* when the number of process groups m is relatively small because the latter chooses systems that allow staging of larger amount of data before their available times in order to guarantee the existence of a solution and the systems with an earlier available time, higher execution rate, and higher duty cycle, could be excluded from the restricted target set [128]. The two versions exhibit similar behavior when $m \geq 50$, but *FIX-DSBAT* does not produce a solution when $m < 50$. When m is large, the

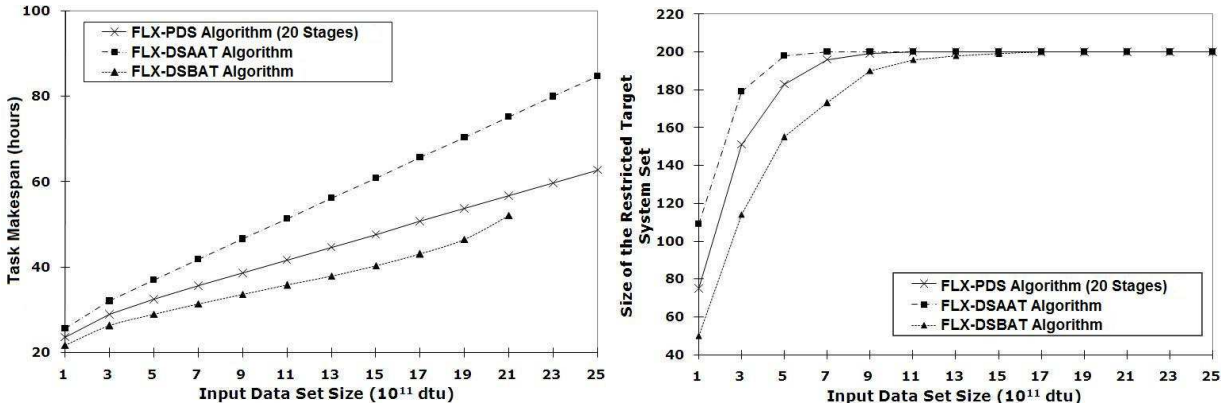


Figure 2.7: The effect of the three different data staging strategies when we allow as many systems as possible to be included in the restricted target set (*FlexMap* scenario). (Left) The task makespan (in hours) function of the input data set size for the *FLX-PDS* with 20 stages, *FLX-DSAAT* and *FLX-DSBAT* algorithms; the input data set size increases in units of 2×10^{11} *dtu*. (Right) The size of the restricted target set (the number of systems used) when the input data set size increases in units of 2×10^{11} *dtu* for the three flavors of the algorithm.

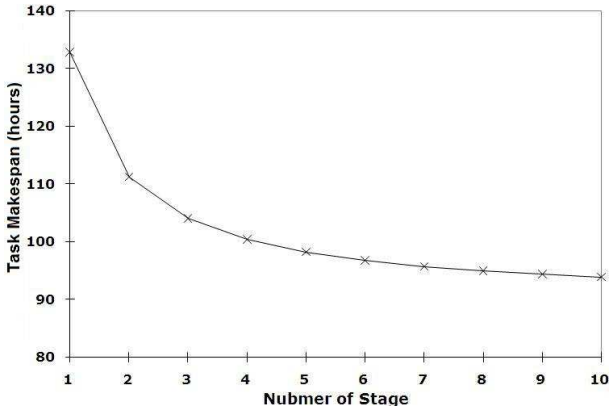


Figure 2.8: Task makespan function of p, the number of pipeline stages for the *FIX-PDS* algorithm; the input data set size is 10^{12} dtu and the number of process groups is 85.

gap between the two algorithms becomes smaller and, eventually, *FIX-DSBAT* outperforms the pipelined version. In order to reduce the task makespan, we can either use more systems, or increase the number of pipeline stages. Indeed, the *FIX-PDS* and *FIX-DSBAT* may lead to the same computation completion time of 54 hours, but require different numbers of target systems, $m = 90$ and $m = 150$, respectively.

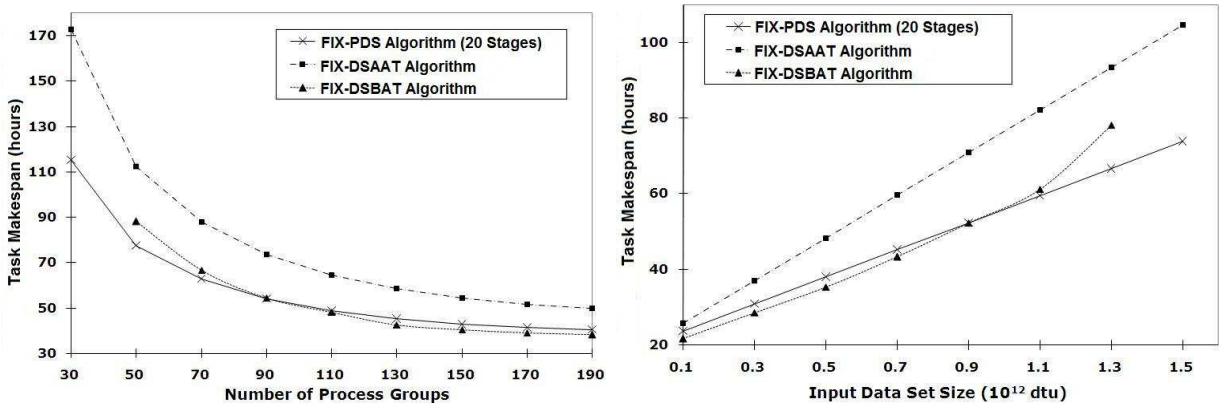


Figure 2.9: The effect of the three different data staging strategies when we limit the number of systems included in restricted target set (*FixMap* scenario). We observe the task makespan for the *FIX-PDS* with 20 stages, *FIX-DSAAT* and *FIX-DSBAT* algorithms. (Left) Function of the number of the process groups; the input data set size is 10^{12} dtu. (Right) Function of the input data set size when the number of process groups is 85.

For a relatively small input data size, *FIX-DSBAT* is slightly better than the pipelined version but it fails to produce a solution when the input data set size is larger than 1.3×10^{12} *dtu*. As the input data set size increases, the gap between the algorithms narrows; when the input data set size is larger than 0.9×10^{12} *dtu*, the pipelined version outperforms the *FIX-DSBAT*. The pipelined version is always better than *FIX-DSAAT* and both produce solutions regardless of the input data set size. We also study the speedup when the size of the process groups m increases but the input data set size is fixed, $\omega = 10^{12}$ *dtu*, for the *FIX-PDS* algorithm. The speedup over different ranges of target systems varies:

$$Speedup_{30-110} = \frac{115}{54} = 2.13$$

while

$$Speedup_{110-200} = \frac{54}{40} = 1.35.$$

Again, the benefits of using an increasingly larger number of systems have to be balanced against the additional cost and overhead to coordinate multiple sites.

2.5.2 Comparative Study of Single- and Multi-port DLS Algorithms

The divisible load model and algorithms introduced in this dissertation are based on a multi-port communication strategy and assume that the coordination site has the capability to distribute the workload in parallel. We expect parallel communication to lead to better performance and in this section we show that the divisible load scheduling model based upon multi-port communication and pipelined execution, referred to as *MultiDLM* outperform the single-port multi-round model, referred to as *SingleDLM*. In case of pipelined execution with ρ stages, we divide the data for each target system into ρ segments of equal size; this allows a system to start the computations as soon as the first segment is received. When $\rho = 1$ we have a single stage execution corresponding to one-round model.

Given n target systems, $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ...\mathcal{S}_n\}$. Let ω_i be the size of the data allocated to system $\mathcal{S}_i$. Call τ_s, τ_m the optimal makespan for the *SingleDLM* and *MultiDLM* models, respectively. First, we assume that all target systems are available at the same time, $\sigma_1 = \sigma_2 = ... = \sigma_n$, as shown in Figure 2.10 for the case $n = 4$. The optimal schedule, the one leading to the shortest completion time, requires that all process groups finish at the same time for both *SingleDLM* [12, 121] and *MultiDLM* (see proof in section 2.4.2).

The divisible load model with one-port communication not only generates the data partitions for each target system, but also computes the data staging schedule, the order in which data is transferred to the individual systems. Without loss of generality, we assume that the data staging in the *SingleDLM* case follows the ordering $\mathcal{S}_1, \mathcal{S}_2, ...\mathcal{S}_n$, Figure 2.10 (a), while in the *MultiDLM* case data staging starts at the same time for all n target systems, Figure 2.10 (b).

In case of *SingleDLM* execution the data staging time and the execution time on each system $\mathcal{S}_i, i \in (1, n)$ are related to the amount of data allocated to the system:

$$\tau_s - \sigma_1 = \frac{\omega_1}{\rho R_1} + \frac{\omega_1}{\mu_1 \eta_1}, \qquad \text{(Eq. 2.30)}$$

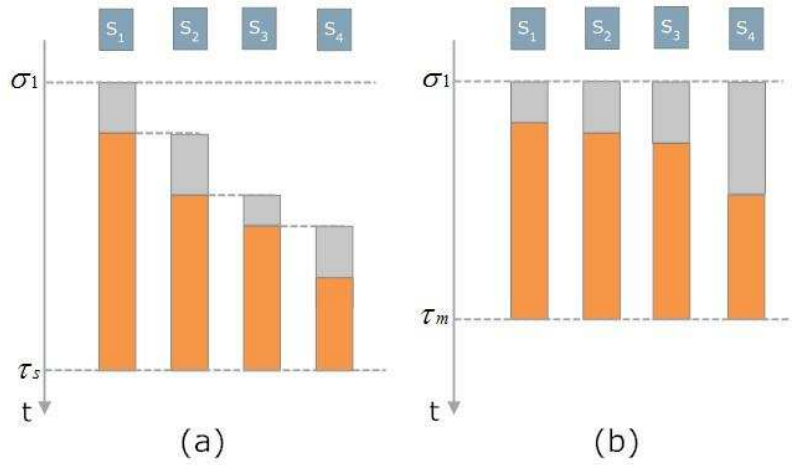


Figure 2.10: Timing diagram showing the data staging and the execution time for $n = 4$ when all target systems start at the same time, $\sigma_1 = \sigma_2 = \sigma_3 = \sigma_4$. The computation on each system starts as soon as the first segment of data is received. (a) *SingleDLM* - divisible load model with one-port communication; τ_s the task makespan. (b) *MultiDLM* - divisible load model with multi-port communication; $\tau_m < \tau_s$ the task makespan.

$$\tau_s - \sigma_1 - \frac{\omega_1}{\rho R_1} = \frac{\omega_2}{\rho R_2} + \frac{\omega_2}{\mu_2 \eta_2}, \qquad \text{(Eq. 2.31)}$$

.....

$$\tau_s - \sigma_1 - \sum_{j=1}^{n-1} \frac{\omega_j}{\rho R_j} = \frac{\omega_n}{\rho R_n} + \frac{\omega_n}{\mu_n \eta_n}. \qquad \text{(Eq. 2.32)}$$

Consider now the *MultiDLM* execution and assume that the amount of data allocated to each system is the same as in the *SingleDLM* case, $\omega_i^m = \omega_i$. If T_i is the process group completion time on the system $\mathcal{S}_i$ then the data staging and the execution times for $\mathcal{S}_i, i \in (1, n)$ are related to the amount of data allocated to the system:

$$T_1 - \sigma_1 = \frac{\omega_1}{\rho R_1} + \frac{\omega_1}{\mu_1 \eta_1}, \qquad \text{(Eq. 2.33)}$$

$$T_2 - \sigma_1 = \frac{\omega_2}{\rho R_2} + \frac{\omega_2}{\mu_2 \eta_2}, \qquad \text{(Eq. 2.34)}$$

.....

$$T_n - \sigma_1 = \frac{\omega_n}{\rho R_n} + \frac{\omega_n}{\mu_n \eta_n}. \quad \text{(Eq. 2.35)}$$

Comparing the *SingleDLM* and the *MultiDLM* executions we notice that:

$$\tau_s - \sigma_1 = \frac{\omega_1}{\rho R_1} + \frac{\omega_1}{\mu_1 \eta_1} = T_1 - \sigma_1, \quad \text{(Eq. 2.36)}$$

$$\tau_s - \sigma_1 - \sum_{j=1}^{i-1} \frac{\omega_j}{\rho R_j} = \frac{\omega_i}{\rho R_i} + \frac{\omega_i}{\mu_i \eta_i} = T_i - \sigma_1, \quad for \;\; 2 \leq i \leq n. \quad \text{(Eq. 2.37)}$$

From Eq. 2.36 and Eq. 2.37 it follows that:

$$\tau_s = T_1, \quad \text{(Eq. 2.38)}$$

$$\tau_s > T_i, \quad for \;\; 2 \leq i \leq n. \quad \text{(Eq. 2.39)}$$

The *SingleDLM* data partitioning may not be the optimal for *MultiDLM*; moreover, the optimal solution for *MultiDLM* execution requires that all process groups finish at the same time, as proved in Section 2.4.2 thus:

$$\tau_s = T_1 > \tau_m. \quad \text{(Eq. 2.40)}$$

We conclude that the optimal solution from *MultiDLM* execution leads to a shorter completion time than the one for *SingleDLM*.

Now we consider the more realistic case, namely that the n target systems have different available times and discuss first the *SingleDLM* case. A *SingleDLM* scheduling algorithm produces in this case a data staging schedule e.g., $\{\mathcal{S}_{i_1}, \mathcal{S}_{i_2}, \ldots, \mathcal{S}_{i_n}\}$ which means that the data staging should first be done for $\mathcal{S}_{i_1}$ and when $\mathcal{S}_{i_1}$ finishes its first segment of data then the data staging for $\mathcal{S}_{i_2}$ could start, and so on. This raises the question whether a system is available or not at the time the data stating schedule allows. We first assume that all systems

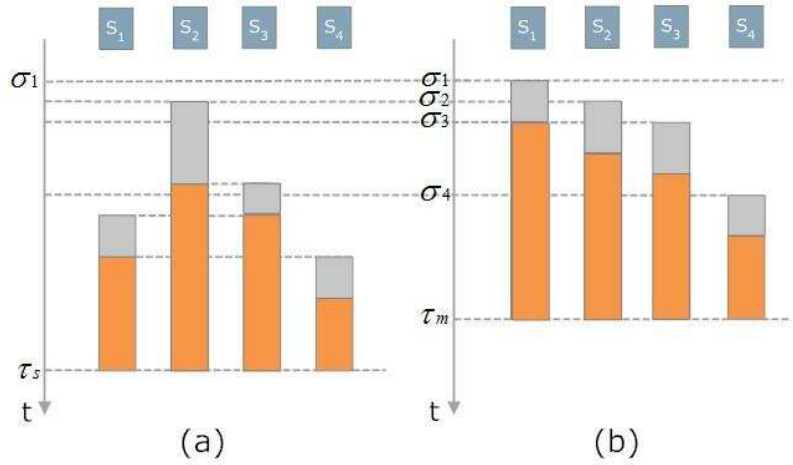


Figure 2.11: Timing diagram showing the data staging and the execution time for $n = 4$ when the available times are different $\sigma_1 < \sigma_2 < \sigma_3 < \sigma_4$ and all systems are available when data staging could begin. (a) *SingleDLM* - divisible load model with one-port communication. The data staging schedule produced by the divisible load algorithm for this example is $\mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_1, \mathcal{S}_4$ thus, data staging on $\mathcal{S}_3$ can only start after $\mathcal{S}_2$ has finished reviving its first chunk of data, and so on. τ_s is the task makespan. (b) *MultiDLM* - divisible load model with multi-port communication. Data staging starts as soon as S_i becomes available at time σ_i and the task makespan is $\tau_m < \tau_s$.

are available at the time when the algorithm allows data staging to begin. For system $\mathcal{S}_i$, we define the system set $\mathcal{P}_i$ which involves all systems before the system $\mathcal{S}_i$ in the sequence of data staging. For example, suppose the data staging schedule on 4 systems are $\mathcal{S}_2$, $\mathcal{S}_3$, $\mathcal{S}_1$, $\mathcal{S}_4$, then $\mathcal{P}_1 = \{\mathcal{S}_2,\mathcal{S}_3\}$. Figure 2.11(a) illustrates the case when the data staging schedule produced by the algorithm is $\{\mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_1, \mathcal{S}_4\}$. In the general case let $\mathcal{S}_a$ be the first system in the data staging schedule. Then we have the following relations:

$$\tau_s - \sigma_a = \frac{\omega_a}{\rho R_a} + \frac{\omega_a}{\mu_a \eta_a}, \qquad \text{(Eq. 2.41)}$$

$$\tau_s - \sigma_a - \sum_{j \subseteq \mathcal{P}_i} \frac{\omega_j}{\rho R_j} = \frac{\omega_i}{\rho R_i} + \frac{\omega_i}{\mu_i \eta_i}, \quad for \;\; i \neq a. \qquad \text{(Eq. 2.42)}$$

Consider now the *MultiDLM* execution and assume that the amount of data allocated to each system is the same as in the *SingleDLM* case, $\omega_i^m = \omega_i$. Then

$$T_i - \sigma_i = \frac{\omega_i}{\rho R_i} + \frac{\omega_i}{\mu_i \eta_i}, \quad for \;\; 1 \leq i \leq n. \tag{Eq. 2.43}$$

From Eq. 2.41, Eq. 2.42, and Eq. 2.43 it follows that:

$$\tau_s - \sigma_a = T_a - \sigma_a, \tag{Eq. 2.44}$$

$$\tau_s - \sigma_a - \sum_{j \subseteq \mathcal{P}_i} \frac{\omega_j}{\rho R_j} = T_i - \sigma_i, \quad for \;\; i \neq a. \tag{Eq. 2.45}$$

A simple derivation leads to:

$$\tau_s = T_a, \tag{Eq. 2.46}$$

$$\tau_s = T_i + \sum_{j \subseteq \mathcal{P}_i} \frac{\omega_j}{\rho R_j} - (\sigma_i - \sigma_a), \quad for \;\; i \neq a. \tag{Eq. 2.47}$$

Consider relation Eq. 2.47, if $\sigma_i \leq \sigma_a$, we will have:

$$\tau_s > T_i, \quad for \;\; i \neq a. \tag{Eq. 2.48}$$

Otherwise, based on the assumption above, because system $\mathcal{S}_i$'s available time is earlier than, or equal to the time when its direct predecessor finishes one round of data staging, it follows that:

$$\sigma_i - \sigma_a \leq \sum_{j \subseteq \mathcal{P}_i} \frac{\omega_j}{\rho R_j}, \quad for \;\; i \neq a. \tag{Eq. 2.49}$$

Thus $\tau_s > T_i$.

The optimal solution for *MultiDLM* requires all process groups to finish computation at the same time thus:

$$\tau_s = T_a > \tau_m. \tag{Eq. 2.50}$$

We conclude that in this case the *MultiDLM* leads to an earlier completion time than *SingleDLM*.

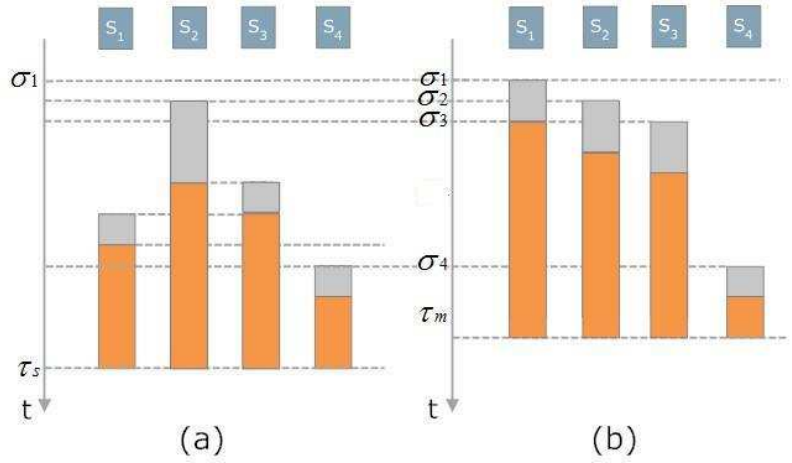


Figure 2.12: Timing diagram showing the data staging and the execution time for $n = 4$ when the avaliable times are different $\sigma_1 < \sigma_2 < \sigma_3 < \sigma_4$ and when some of the systems become available after the data staging could start. (a) *SingleDLM* - divisible load model with one-port communication. The data staging schedule produced by the divisible load algorithm is $\mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_1, \mathcal{S}_4$ and the data staging on S_3 can only start after the one for S_2 has finished, and so on. Note that $\mathcal{S}_4$ is available at time σ_4 though the data staging could have started earlier. τ_s is the task makespan. (b) *MultiDLM* - divisible load model with multi-port communication; data staging starts as soon S_i becomes available at time σ_i. The task makespan is $\tau_m < \tau_s$.

Lastly, we examine the case when one or more systems are not available at the time when data staging could start; obviously, this may only occur in the *SingleDLM* execution. Figure 2.12 (a) illustrates the case when S_4 is available at time σ_4 though the data staging could have started earlier.

Under the *SingleDLM* algorithm, the relationship Eq. 2.41 still holds for the first system, $\mathcal{S}_a$ in the data staging schedule. Call $\mathcal{Q}$ the subset of target systems whose available times are later than the data staging completion times of their predecessors in the data staging schedule. For systems in $\mathcal{Q}$, we have:

$$\tau_s - \sigma_i = \frac{\omega_i}{\rho R_i} + \frac{\omega_i}{\mu_i \eta_i}, \quad for \;\; \mathcal{S}_i \subseteq \mathcal{Q}. \qquad \text{(Eq. 2.51)}$$

For the other systems, we obtain a relation similar to Eq. 2.42:

$$\tau_s - \sigma_a - \Phi_i = \frac{\omega_i}{\rho R_i} + \frac{\omega_i}{\mu_i \eta_i}, \quad for \;\; \mathcal{S}_i \subseteq (\mathcal{S} - \mathcal{Q} - \mathcal{S}_a), \tag{Eq. 2.52}$$

where Φ satisfies:

$$\Phi_i \geq \sum_{j \subseteq \mathcal{P}_i} \frac{\omega_j}{\rho R_j}. \tag{Eq. 2.53}$$

As $\mathcal{P}_i$ is the set of predecessors of $\mathcal{S}_i$ in the data staging schedule, Φ_i is equal to $\sum_{j \subseteq \mathcal{P}_i} \frac{\omega_j}{\rho R_j}$ if no system is in both $\mathcal{P}_i$ and $\mathcal{Q}$. Otherwise, there will be at least one system which has to wait for its available time before it can receive the data and thus "postpone" the actual available time of system $\mathcal{S}_i$.

The relationship Eq. 2.43 still holds for all systems; from the relationships Eq. 2.41,Eq. 2.51, Eq. 2.52, and Eq. 2.43, we can derive Eq. 2.50 to conclude the proof.

2.6 Design of a Divisible Load Meta-scheduler

In this section we describe the architecture of the DLS meta-scheduler and outline the interactions between the meta-schedulers on the coordinator site and the target systems during the task submission process.

2.6.1 DLS Meta-scheduler

The DLS system is implemented as a peer-to-peer system [100]. The DLS code is replicated on all target systems and each system can play the role of coordinator or execution engine. Figure 2.13 shows the four components of the DLS meta-scheduler: the *Resource Manager (RM)*, the *Monitor (M)*, the *Scheduler (S)* and the *Execution Manager (EM)*. Each system supports bi-directional communication interfaces.

The *Resource Manager* initiates the negotiation between the DLS meta-scheduler on the coordinator site and the DLS meta-schedulers on multiple target systems. Its primary role is

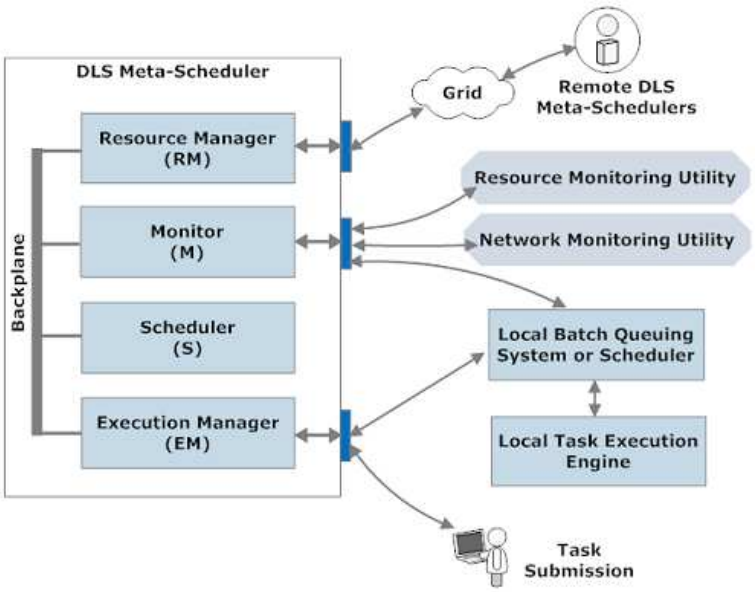


Figure 2.13: The architecture of the DLS meta-scheduler with the interface to interact with the external services and devices.

to generate the target systems set. The *RM* integrates the security authentication, message communication handling, and the data management functions for the target systems pool. It operates by listening for incoming messages from the remote resources and responding to remote resources on the specific ports through its external interface. The *RM* plays also the role of a message dispatcher and relays messages to the appropriate units.

The role of the *Monitor* is to report on the status of local resources. When a system joins the potential target systems set, the local *Monitor* supplies the coordinator site with information regarding the status of local resources, i.e., the performance evaluators. The *Monitor* uses several utilities to gather the necessary data: (i) the *Historical Execution Data Analyzer* to predict the execution rate; (ii) network utilities such as *Network Weather Service (NWS)* [119] and *iperf* [52] to measure the data transfer rate; (iii) the local batch-queuing system or local scheduler such as *PBS* [87], *Condor* [105], *Sun Grid Engine* [103], and *WebCom* [85] to estimate the available time and duty cycle.

The *Scheduler* executes the DLS algorithms. It generates the mapping, stores it in the Schedule Table and informs the *Resource Manager* to send resource reservation messages to the selected sites.

The *Execution Manager* processes task submission requests and responses to the task execution status queries from the user. It also acts as the interface between the DLS meta-scheduler and the local batch-queuing system or local scheduler on the same site. It can make resource reservation, submit the task instance and monitor the execution by inquiring the local system, and handle the task instance execution exceptions.

2.6.2 Task Submission in DLS System

Figure 2.14 illustrates the interactions between DLS meta-schedulers on the coordinator site and on the target system equipped with execution engine. A DLS session is initiated when the *Execution Manager* receives a `Task Submission` request from a user; this site will play the role of the coordinator for this transaction. The *EM* analyzes the request and identifies critical information regarding the task description, the hardware/software requirements, the QoS requirements, and location of input data. Then it passes this information to the local *RM* which, in turn, initiates negotiations with the *RMs* on the other peers. The information in the task submission request and the identity of the coordinator are included in a `Task Execution` request sent by the local *RM* to the *RMs* of the remote peers.

Upon receiving a `Task Execution` request the *RM* of a peer collaborates with the local *Scheduler* to determine if the requirements for the given task can be fulfilled. For instance, it checks if the software environment is properly set and if a stable network connection to the input data repository exists. If all the requirements can be satisfied, the local *Monitor* predicts the performance evaluators for the given task and informs local *EM* to make a temporary resource reservation; the temporary resource reservation will be canceled if the reservation is not confirmed by the coordinator within a timeout period. Then a `Request`

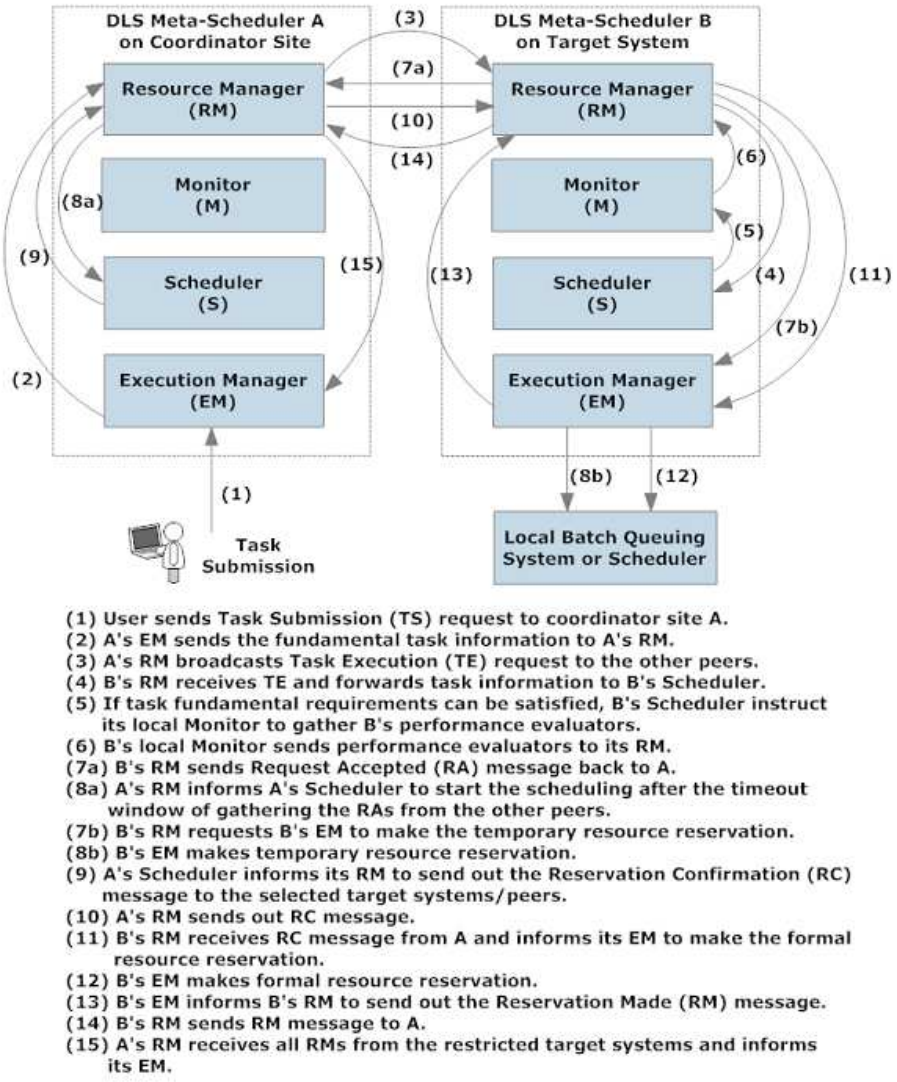


Figure 2.14: Task Submission in DLS Scheduling System.

`Accepted` message which includes the identity of the target system, the value of each performance evaluator returned by local *Monitor* and the basic hardware/software setup are sent back to the coordinator.

The *RM* on the coordinator site receives `Request Accepted` messages and creates the target systems set $\mathcal{S}$. After a certain timeout period (it is shorter than the timeout period for the temporary resource reservations on the target systems), the coordinator will refuse new incoming `Request Accepted` messages and will inform the *Scheduler* to carry-out the

DLS algorithm. A set of target systems forming the restricted target set will be selected by DLS algorithm from $\mathcal{S}$ to run the task. Then the *RM* on the coordinator site sends the `Reservation Confirmation` message to the peers in the restricted target set.

As soon as the *RM* of a peer receives the `Reservation Confirmation` message it informs the local *EM* to turn the temporary resource reservation into a permanent one. A `Reservation Made` acknowledgement is sent back to coordinator to confirm a successful reservation.

After receiving all `Reservation Made` acknowledgements, the coordinator finalizes the schedule/mapping and starts the task execution and monitoring; its *EM* supervises the execution of the task and periodically reports the task status, takes appropriate actions when exception occur, forces the termination of the DLS session if necessary, and closes the DLS session when the task finishes.

CHAPTER 3: SELF-ORGANIZING LARGE-SCALE DISTRIBUTED SYSTEMS

In this section we first survey a model suitable for the study of large-scale distributed system and then apply the model to a case study.

3.1 A Framework for Modelling Self-organizing Systems

The current state of the art with respect to system complexity and self-organization and the highly abstract concepts developed in the context of natural sciences [91] do not lend themselves to design principles for engineering, self-organizing computing and communication systems. In response, we attempt to map the attributes of a self-organizing system into an actionable modelling framework; *actionable* means that the resulting model can guide the design of practical systems.

First, the model initially described in [76] is refined to capture the nondeterministic behavior of large assemblies of entities. To achieve this goal we use a metaphor from quantum mechanics [36]. The formalism we use, and discussed in detail elsewhere [77] is complex; here we sketch the basic tenants below. An entity is characterized by a collection of properties, or genes; a gene captures the role played by the corresponding property in any interaction involving the entity. The genes can be classified in several categories, including *makeup* genes describing the components of an entity, *decision* genes which affect the path taken when multiple alternatives are available, *action* genes controlling the set of actions an entity may undertake, and *specificity* genes such that some are specific to an entity, others to a class of entities, or to all entities.

A gene $\bar{g}$ is represented as a vector in $\mathcal{H}_2$, a two-dimensional vector space over the field of complex numbers, $\mathbb{C}$. For example, $\bar{g}_0$, $\bar{g}_1$, and $\bar{g}$ are genes that affect the behavior with probabilities $0, 1$, and $0 \leq p_1 =\mid g_1 \mid^2 \leq 1$, respectively:

$$\bar{g}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (1,0)^T, \qquad \bar{g}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = (0,1)^T \qquad and \qquad \bar{g} = \begin{pmatrix} g_0 \\ g_1 \end{pmatrix} = (g_0, g_1)^T. \tag{Eq. 3.1}$$

In the last expression, $g_0, g_1 \in \mathbb{C}$ and $\mid g_0 \mid^2 + \mid g_1 \mid^2 = 1$. The probability that the gene $\bar{g}$ does not affect behavior is $p_0 = \mid g_0 \mid^2$. Genes can be used to express nondeterministic behavior, for example, the probability of a successful transmission on a broadcast channel. Genes can also be used to express the relative weight of different factors affecting the behavior of an entity.

Two genes are orthogonal if their inner product is equal to zero, e.g., $\bar{g}_0 \cdot \bar{g}_1 = 0$; this means that the two properties of the entity associated with these genes are independent of each other. The *genetic sub-state* of a set of n related genes of an entity is the tensor product of the corresponding genes, a vector in a 2^n dimensional space. For example, the genetic communication sub-space of a sensor is spanned by four genes; there are 2^4 complex numbers describing the communication behavior of a sensor during the self-organization phase. The moduli of these complex numbers are probabilities of different communication events.

Borrowing terms from structural biology we envision the formation of primary, secondary, or higher level structures of increased complexity based upon the genetic state of entities. An *affinity/binding* function [110] uses the genetic state, or genetic sub-states, to determine the degree of compatibility among entities when they form such structures. An affinity function could use not only the projections of a vector on a sub-space, e.g., the moduli of the 16 complex numbers in the previous case, but also their phases to establish the required degree of matching of the properties expressed by the genes. For example, a phase of $0 \leq \phi \leq \pi/4$ could mean perfect matching of the projections, while $\pi/4 < \phi \leq \pi/2$ could tolerate a 20% difference.

Actions have genetic side-effects, they transform the gene of the system; the transformation of n genes is described by a $2^n \times 2^n$ matrix; a controlled operation transforms one or more *target* genes depending on a *control* gene. For example, I, X, and Z transform a single gene, while a $CNOT$ transforms a gene called a *target* depending on the value of another one, called a *control* gene:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad and \;\; CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{Eq. 3.2}$$

This approach not only allows the model to capture non-deterministic behavior and genetic evolution as a result of actions involving the entity, but also the interactions of an entity with the environment. The model highlights the explosion of the genetic state space when the number of genes increases. To deal with the explosion of the genetic state space we can model an entity at various levels of detail or focus on a particular aspect of the behavior described by a subset of genes and represented as a vector in a genetic subspace; for example we can focus on the communication activities of a sensor and ignore the data collection activities. It also supports *macroscopic characterization* of entities based on a very large number of microscopic attributes; for example, we can use the entropy as a macroscopic measure of the genetic state. The model is able to describe composite entities using the density matrix [78] and to estimate the properties of a composite entity by the partial trace of the density matrix [78]. The question we plan to address in the future is if, in addition to its theoretical appeal, this formalism could be useful in practice to guide the modelling effort.

In addition to genes, an entity could have *intrinsic properties* such as identity, age, total energy, the seeds for the random number generators used by the sensors of a Very Large

Sensor Network (VLSN) to decide when to transmit and what radio frequency to use, and so on.

The *environment* mediates the interactions among entities and plays a critical role in the dynamics of self-organization described by our framework. The description of the environment includes the elements shared by all entities, including the public and private channels used by entities to communicate, the actions feasible in the system, as well as, the standard energetic side-effects of each action. Though not fully explored in this dissertation we believe that the introduction of the environment as a critical component of self-organization opens the possibility to model effectively non-linear effects, and study avalanche phenomena as well as phase transitions.

The energetic side-effects of the actions of an entity are captured by the following cycle: initially, when an entity is created it is provided with a certain amount of *potential energy* from a common pool (the makeup gene controls the actual type and amount of resources the entity can convert the potential energy into); to carry out an action an entity consumes *kinetic energy*; the successful completion of an action rewards the entity with *effective kinetic energy* that can be transformed in kinetic, potential, and even stored for future use. The potential energy is a metaphor for the resources e.g., CPU cycles, memory, power, and so on, required to carry out the specific actions or transactions the entity is capable to perform. The effective kinetic energy is provided by the environment as a side effect of a successful transaction and, in turn, the environment recovers it from the other entity or entities involved in the transaction. This distinction between the three forms of energy is inspired by thermodynamics; it captures the need to transform one form of energy into another as well as the efficiency of the energy conversion process. It also offers entities the flexibility to store effective kinetic energy and to convert it according to the current needs. It motivates the entities to interact, an idle entity cannot accumulate effective kinetic energy thus, it is not capable to adapt to new conditions.

Entities communicate using epidemic/gossiping algorithms [33]. All entities e_i in the universe $\mathcal{U}$ share a *public communication channel* $\mathcal{C}$. Information broadcast by an entity e_i over this channel is received by a limited subset of entities and may be disseminated by them using *gossiping/epidemic algorithms*; there is no guarantee that information broadcast by an entity e_i will reach all entities in $\mathcal{U}$. Each message m has a time to live, τ_m, expressed in number of hops in the gossiping algorithm. Concurrent communication is possible because of the limited range of individual broadcasts, but there is no guarantee that two simultaneous broadcasts will not interfere with each other.

To prevent overloading the channel $\mathcal{C}$, entities have the option, and the motivation, to communicate through *private communication channels*, $\mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(p)}$. For example, the goal of the self-organization phase of the VLSN is the establishment of private communication channels; these private communication channels are collision-free wireless connections allowing the sensors to operate with minimal energy consumption during the activity phases. Broadcasting a message on the public communication channel $\mathcal{C}$, or on the private communication channel $\mathcal{C}^{(p)}$, requires consumption of kinetic energy. An entity retransmits messages from its neighbors on either the public or a private channel.

The formalism we just described is at the core of self-organization when decisions are based solely on local information and permits the formation of stable non-equilibrium structures, but it cannot account for the non-linear behavior. Our search for a solution to this problem was guided by an insight due to Alan Turing: in his seminal paper on morphogenesis [108], he suggests that a mechanism based on slow diffusion of an activator and fast diffusion of an inhibitor can lead to *formation of stable stationary non-equilibrium patterns through the diffusion of some key compounds.*

Our approach to model non-linearity is based on activator and inhibitor mechanisms. The activator is the reward for an action, while the inhibitor is a mechanism to modulate the reward based on the *reputation* of an entity defined in Equation Eq. 3.4. A reward reinforces behavior; when offered, it causes a behavior to increase in intensity. The reputation reflects

the level of satisfaction, how well the results match the individual expectation of each entity involved in a transaction.

Different functions can be used to model the reward and we suggest the use of a sigmoid [6]:

$$\chi(n) = \frac{(n/\omega)^{\zeta}}{1 + (n/\omega)^{\zeta}} \leq 1 \qquad \text{(Eq. 3.3)}$$

where ζ and ω are constants with $\zeta \geq 2$, and $\omega > 0$; then $\chi(\omega) = 1/2$, Figure 3.1. The sigmoid is often used to model the evolution of a biological system from birth to maturity and demise, the three stages clearly identifiable in Figure 3.1 where the argument n reflects the number of actions in the current epoch. An epoch starts when an entity joins an existing structure or initiates the creation of a new one. When the entity dissociates itself from a structure the count of the number of actions or transactions is reset to zero. We believe that an important trait of self-organization is the ability of a viable structure to transition from one sigmoid to another at an optimal time, in order to sustain optimal behavior. In this case an epoch may consist of several sub-epochs; for example, Figure 3.1(b) illustrates the case when an epoch consists of three sub-epochs, the first one corresponds to evolution following sigmoid S_1, the second one follows S_2, and the third S_3. This type of evolution is evident in many social and economical systems.

The reputation of an entity plays an important role in evaluating the actual reward for the entities involved in a transaction. The main idea of the algorithm is to allow an entity to enhance its reputation gradually in the absence of negative feedback and to heavily penalize reputations in the light of negative feedback. In the absence of negative feedback the reputation increases exponentially at first, then experiences a linear increase. Negative feedback drastically lowers the reputation. The feedback is provided to the partner of a transaction after the completion of the transaction and it may force an entity to re-adjust the reputation of the partner.

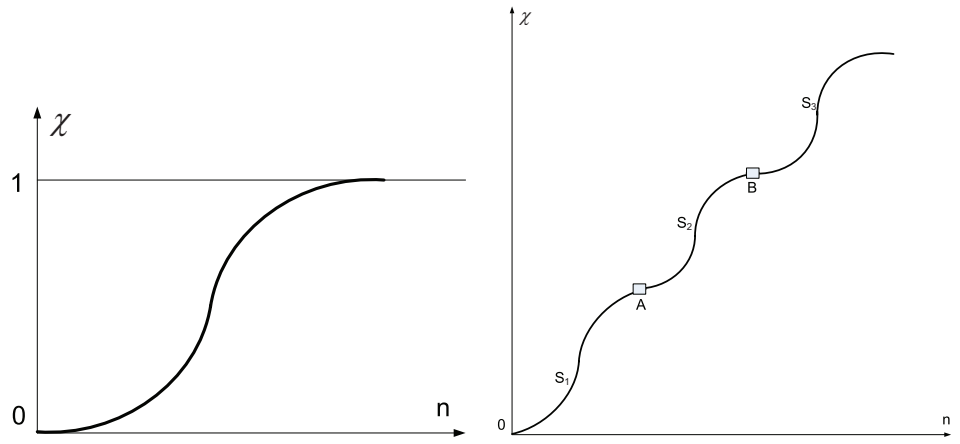


Figure 3.1: (a) A sigmoid is used to model the reward $\chi = \chi(n_i)$ for an entity e_i. (b) The behavior of a viable structure able to transition from one sigmoid to another at an optimal time, in order to sustain optimal behavior. Three sigmoids S_1, S_2 and S_3 are shown; the transition from sigmoid S_1 to sigmoid S_2 occurs at instance A and the one from S_2 to S_3 at B.

Let $\mathcal{R}_j^i(n)$ be the reputation of entity j evaluated by entity i after completion of n transactions. Initially $\mathcal{R}_j^i(0) = 0$. In the absence of negative feedback the reputation increases exponentially at first and subsequently experiences a linear increase:

$$\mathcal{R}_j^i(n) = \begin{cases} \kappa_1 2^n & \mathcal{R}_j^i(n) \leq \mathcal{R}_t \\ \min\left[k_2 n, \mathcal{R}_h\right] & \mathcal{R}_t < \mathcal{R}_j^i(n) < \mathcal{R}_h \end{cases} \qquad \text{(Eq. 3.4)}$$

with n the number of transactions, k_1, k_2, $\mathcal{R}_t$ and $\mathcal{R}_h$ constants specified by the rewards gene. There are two thresholds $\mathcal{R}_t$ and $\mathcal{R}_h$; the reputation increases exponentially until the $\mathcal{R}_t$ threshold is reached, after that it increases linearly until it reaches the upper limit of $\mathcal{R}_h$. The negative feedback lowers the reputation to 0 if it is below the threshold $\mathcal{R}_t$. The reputation drops to $\mathcal{R}_t$ any time negative feedback is received after this threshold is reached:

$$\mathcal{R}_j^i(n+1) = \begin{cases} 0 & \textit{if} \quad 0 < \mathcal{R}_j^i(n) < \mathcal{R}_t \\ \mathcal{R}_t & \textit{if} \quad \mathcal{R}_t \leq \mathcal{R}_j^i(n) \leq \mathcal{R}_h. \end{cases} \qquad \text{(Eq. 3.5)}$$

As the systems we are concerned with are deployed and are under development at the same time, self-organization should create symbiotic clusters consisting of "old" and "new" structures with the same functionality and allow the "old" ones to disappear only after the functionality of the "new" ones has been thoroughly tested. This process can be controlled by an aging gene, so that the aging process is slow when a new structure appears and it is accelerated as the structure matures.

3.2 Case Study: Pleiades, A Self-Organizing Resource Sharing System

In [76] we discuss a market-centric implementation of a Pleiades system; in this dissertation we consider an implementation based entirely on self-organization. The infrastructure enables interested parties to form organizations grouping together entities based upon performance metrics such as: security, quality of service (QoS), fault-tolerance, timing constraints, or carbon footprint.

The entities in our generic model are computer systems and the structures are called *Virtual Organizations* (VO); VOs are formed dynamically, and may be disbanded and reorganized, as members join and leave. A majority of transactions are carried out between the members of the same VO. An entity which has not yet joined any organization is called a *free agent.* A free agent aims to join a VO to optimize its state in the goal subspace. The interactions among entities in VO are based on genetic information and are limited by imprecise knowledge of the global state. VOs allow members to act effectively on local information and pursue self-sufficiency.

In our model we first describe the environment as well as the genetic makeup of individual entities. The environment specifies the set of services available, as well as, the kinetic energy required for a transaction involving a given service and the standard reward per transaction.

The environment also describes the communication channel and its properties. For example, the environment specifies the probability of a successful transmission on the public

channel. The actions in this environment are: an invitation to join a VO (such an invitation can only be issued by a service provider); acceptance of an invitation to join a VO; re-broadcasting an invitation; a request from a service provider for acquiring resources, a reply to such a request, an offer to provide resources, a reply to such an offer.

In this initial study we have not modelled the non-linear behavior; instead the reward mechanism favors activities with positive social impact. During the self-organization phase the only activities are requests to form and to join a VO sent over the public channel with a probability of success $p = 0.8$; the standard reward for sending such a request increases the effective kinetic energy by a small amount, say 2 units. Communication over the private channel increases the kinetic energy by 1 unit with a probability of success $p = 0.9$. Successful formation of a VO increases the kinetic energy of the entity starting the VO and of the entities joining the VO by 10 units and also adds 8 units of effective kinetic energy.

An entity is a computer with a genetic state consisting of several subspaces including resources, performance, entity role, actions, energy conversion, and goals. The genes in each subspace describe respectively: (i) the "active" material, or resources such as, CPU cycles, main memory, secondary storage, graphics capabilities, or network bandwidth; (ii) the performance metrics such as quality of service (QoS), security, reliability, and carbon footprint; (iii) the function, of the entity; a service provider and consumer of resources, or a client of services and provider of resources; (iv) the entity-specific actions, e.g., the service provided by a service provider; (v) the efficiency of converting potential into kinetic energy, and effective kinetic into potential energy; (vi) the entity-specific, class, and common goals; (vii) entity-specific genes.

Initially, an entity is allocated an amount of potential energy from a common pool; the resource genes of the entity define the relative percentage of the energy allocated for each type of resource and operation mode, as well as, the energy conversion factors. The model is able to capture more intricate aspects of resource management: the consumption of one type of resource affects other resources; some resources are perishable, if not used they are wasted and

there is a cost associated with idle resources. For example, consider an entity granted initially 1000 *eu (energy units)* and having only two genes $\bar{g}_c = (g_{c0}, g_{c1})^T$ and $\bar{g}_m = (g_{m0}, g_{m1})^T$ the first describing the computing power and the second the memory; the four projections on the subspace spanned by the two genes are: $\mid g_{c0}g_{m0} \mid^2, \mid g_{c1}g_{m0} \mid^2, \mid g_{c0}g_{m1} \mid^2$ and $\mid g_{c1}g_{m1} \mid^2$. These projections describe respectively the amount of energy allocated for: (i) the idle mode (none of the two perishable resources are in use); (ii) the mode when the CPU is used intensively and memory occasionally (compute-intensive processing); (iii) the mode when the memory is used intensively (memory-intensive processing); and, finally, (iv) the mode when both are used intensively. When $g_{c0} = g_{m0} = g_{c1} = g_{m1} = 1/\sqrt{2}$ then the entity is allocated 250 eu for each one of the four modes of operation. A more simplistic model supports only the last mode of operation when both resources are used intensively; then $g_{c0} = g_{m0} = 0$ and $g_{c1} = g_{m1} = 1$. Recall that $g_{c0}, g_{m0}, g_{c1}, g_{m1}$ are complex numbers and that $\mid g_{c0}g_{m0} \mid^2 + \mid g_{c1}g_{m1} \mid^2 + \mid g_{c0}g_{m1} \mid^2 + \mid g_{c1}g_{m1} \mid^2 = 1$.

The performance subspace could be spanned by several performance genes; here we consider three genes $\bar{g}_q, \bar{g}_s$ and $\bar{g}_r$ for quality of service, security, and reliability. In this case the performance is a vector in a 2^3-dimensional space. Then, the relative weight of the quality of service, security, and reliability will be the products of complex numbers describing individual genes for either a service provider, or a resource provider: $\mid g_{q1}g_{s0}g_{r0} \mid^2, \mid g_{q0}g_{s1}g_{r0} \mid^2$, and respectively, $\mid g_{q0}g_{s0}g_{r1} \mid^2$, with $\mid g_{q1}g_{s0}g_{r0} \mid^2 + \mid g_{q0}g_{s1}g_{r0} \mid^2 + \mid g_{q0}g_{s0}g_{r1} \mid^2 = 1$. Other approaches for modelling the performance metrics are possible, for example a service provider may use all 8 (eight) projections to quantify different mixes of measures. The reputation parameters encode the following values $k_1 = 1$ and $k_2 = 2$; the ones of the reward function are $\zeta = \omega = 2$.

Most random variables, including those representing the amount of potential energy distributed to individual entities are modelled using heavy-tail distributions, probability distributions whose tails are not exponentially bounded. We believe that properties of the systems we investigate, such as heterogeneity, diversity, and dissimilarity justify the choice

of this approach which is also supported by the self-similarity, a behavior observed in the Internet traffic [116].

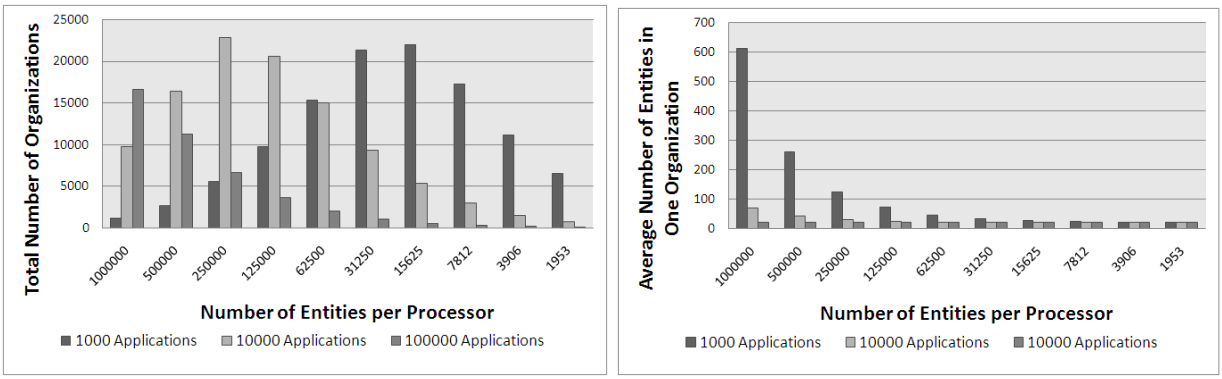
The focus of our study is the formation of VOs and resource management within a VO; the model is designed to capture relevant performance data related to these objectives rather than the communication aspect of self-organization. We only mention that an organization initiating the formation of a VO includes in its invitation the identification of the private channel used for communication by entities joining the VO. The details of the communication protocols and the format of messages can be found elsewhere [77]. Here we only mention that a request for resources includes the performance genes as well as the gene describing the active material and the energetic makeup. The energetic makeup in this case is based upon the assumption that the kinetic energy consumed by an entity in a transaction is replenished and additional effective kinetic energy rewards a useful activity. The *effectiveness* of an entity is then measured by the ratio effective kinetic energy to kinetic energy.

3.3 Evaluation of a Pleiades Model Through a Simulation Study

First, we investigate the dynamics of VO formation, the relationship between the number of VOs and the number of entities in one VO for a fixed size of entity population. We study the answers to several questions [77]: Is the self-organization model scalable? What is the effect of the interplay of the population size and genetic diversity on the model? Do our metrics of optimality lead to stable non-equilibrium patterns?

The simulation consists of three steps: (i) the creation of entities; (ii) the self-organization phase; and (iii) the activity phase. The entity initiating the formation of a VO is a service provider which selects an application and invites other entities to join that VO. Transactions involving service providers and clients which are at the same time resource providers take place during the activity phase.

The generation of entities and the formation of VOs is done in parallel; the common pool of entities and active material is divided into P groups and N entities are created in parallel using P processors. A *group* of $n(P) = N/P$ entities are allocated to each system. We consider a fixed number $N = 10^6$ of entities; P, the number of groups/processors, ranges from 1 to 512, and we experiment with $1,000, 10,000$, and $100,000$ services. Call $n(P)$ the size of the population of one group when there are P groups; $n_{SP}(P), n_{RP}(P)$ and $n_{Serv}(P)$ are the number of service providers, resource providers, and services, respectively.

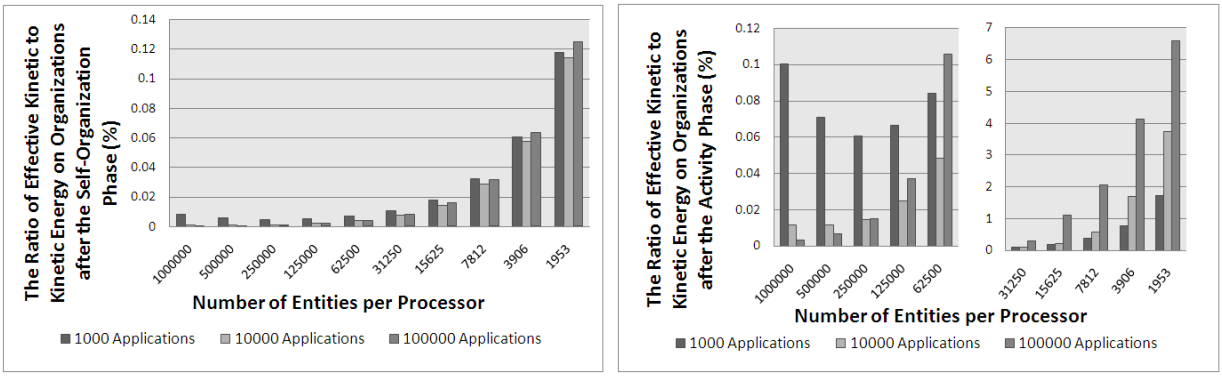


(a) (b)

Figure 3.2: (a) The number of VOs function of the number of entities in each group. (b) The average size of a VO.

The self-organization phase. First we report on the effect of population size and the genetic diversity. Figures 3.2(a) and 3.2(b) show that the number of services plays a critical role in determining the number of VOs and the average size of a VO for a given size of the entity population, N. Indeed, when there is only one group $n(1) = 10^6$ and $n_{SP}(1) \approx n_{RP}(1) \approx 500,000$, and there are some $1,000$ services then we have slightly more than $1,000$ VOs and the average number of entities in one VO is slightly larger than 600. This result is encouraging, it shows that the algorithm creates redundant structures, in other words the service providers of the same service are likely to join the same VO and if one fails, the same service can be provided by others.

When the number of services increases by two orders of magnitude, $n_{Serv}(1) = 100,000$ and the number of number of service providers and resource providers are $n_{SP}(1) \approx n_{RP}(1) \approx 450,000$ then the number of VOs is much larger, around $17,000$ and the number of entities in each VO is around 35. As we decrease the group size by increasing the number of groups from 1 to $2, 4, \ldots, 512$ we end up with smaller and smaller group sizes and the effect of the number of services changes. For example, when the group size is $n(32) = 31,250$, and the total number of services is $1,000$, we have $n_{SP}(32) \approx n_{RP}(32) \approx 16,000$. The total number of VOs for all 32 groups is about $22,000$, so each group had roughly $22,000/32 \approx 680$ VOs and an average a VO has about 30 entities. When the total number of services is $100,000$ the number of VOs per group is much smaller, in the low teens.



(a) (b)

Figure 3.3: The ratio of effective kinetic energy to kinetic energy. (a) After the self-organization phase. (b) After an activity phase consisting of 10^5 transactions.

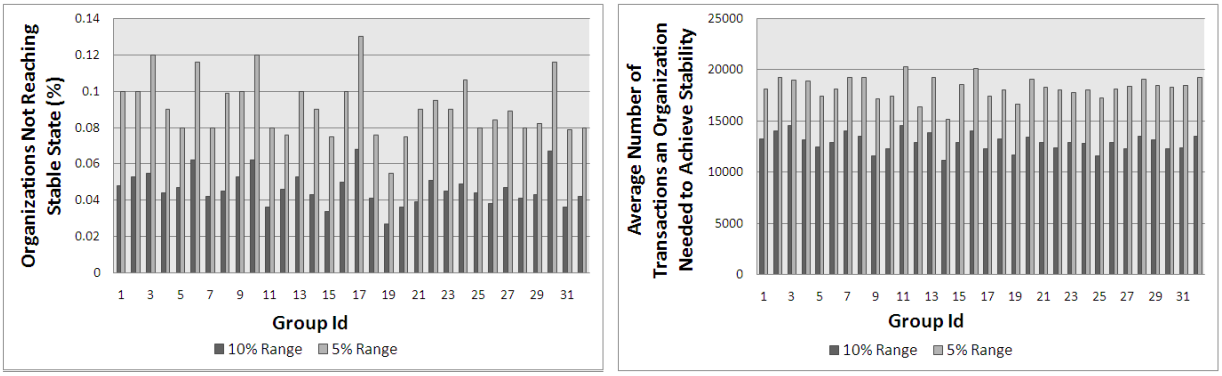
Figure 3.3(a) shows that the size of the population has a significant effect on the energy spent for VO formation. The ratio of effective kinetic to kinetic energy is abysmal, lower than 0.01%, when the population size is very large, (10^6 entities), and increases 10 times to 0.1% when the population size decreases by a factor of 512. Figure 3.3(b) shows the ratio of effective kinetic energy to kinetic energy for an activity phase consisting of 10^5 transactions. The number of transactions per VO is adequate when we have fewer than 100 VOs; in this case each VO experiences on average $1,000$ transactions. Indeed, this ratio has reasonable

values, about 6.5% only for the small population size, $n^{512} = 10^6/512$ with 100,000 services. We conclude that *the population size and the genetic diversity are critical for the model* and a simulation study to determine optimal values for these parameters of the model should be conducted before a more intricate analysis of the self-organizing system is done.

Now we discuss the stability of the structures. We consider the configuration corresponding to $1,000$ services and two groups thus, $500,000$ entities per group, about 3000 VOs/group, and an average of about 270 entities per VO. We construct 32 replicas of such groups, each group with a different number of VOs, average number of entities in one VO, and entities in each VO. Our results show that more than 30% of all VOs reach a ratio of effective kinetic energy to kinetic energy better than or equal to 50% after 25,000 transactions. Only 13% of all VOs reach a ratio of 0.25% or less. We monitor these ratios after each batch of 500 transactions per group and consider that a VO has reached a *stable and optimal* activity level when the average ratio for the VO does not fluctuate during 50 consecutive batches outside a band of width $\pm\delta$. In other words, the difference between the maximum and minimum ratio for 50 consecutive measurements is not larger than 2δ. In our experiments, we consider two values, $\delta = \pm 5\%$ and $\delta = \pm 10\%$. A VO whose average ratio after $25,000$ transactions does not satisfy this criteria is considered unstable and incapable of achieving optimal behavior.

Figure 3.4(a) shows that for $\delta = \pm 10\%$ only a very small fraction, $0.04 - 0.06\%$, of VOs are unable to achieve optimal and stable behavior and reach stable and optimal behavior after approximately $10,500$ to $14,500$ transactions; when $\delta = \pm 5\%$ this fraction increases to $8 - 12\%$ and $15,000$ to $20,000$ transactions are needed to reach equilibrium, Figure 3.4(b).

The activity phase. A more realistic scenario is that, n_{SP}, the number of service providers is much larger than n_{RP}, the number of resource providers. In our experiments, $n = n_{SP} + n_{RP} = 10^6$, there are relatively few service providers $1,000 \leq n_{SP} \leq 5,000$ and a large ratio, $n_{RP}/n_{SP} \approx 1,000$.



(a) (b)

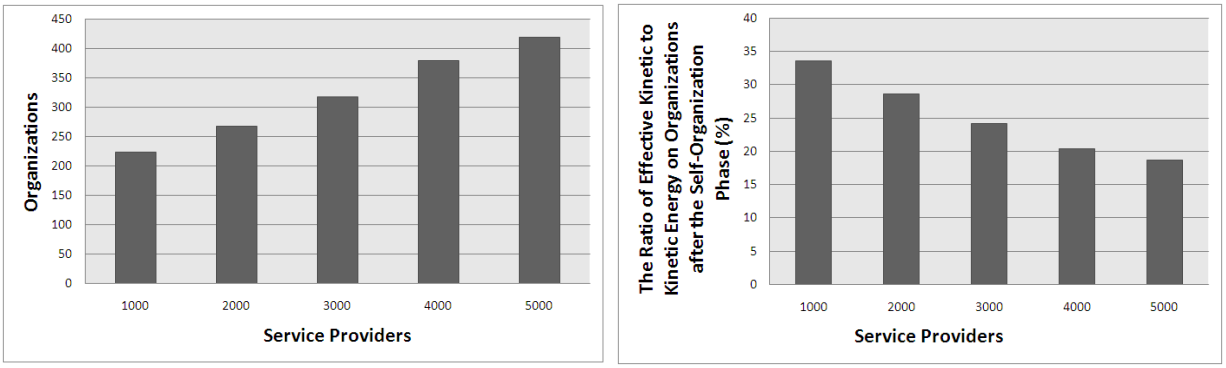
Figure 3.4: (a) Fraction of VOs unable to achieve stable behavior in each of the 32 groups with $\delta = 5\%$ and $\delta = 10\%$. (b) Average number of transactions needed to achieve stable behavior in each of the 32 groups when $\delta = 5\%$ and $\delta = 10\%$.

Once a VO is formed, the service providers start acquiring resources based on the characteristics of the service and the expected load, modelled as an intrinsic property of the entity. A transaction has a failure probability of $p_{fail} = 0.1$. After each set of $1,000$ transactions all VOs and all entities in a VO evaluate the ratio of effective kinetic to kinetic energy. Entities unable to increase their effective kinetic energy to kinetic energy ratio by at least τ_{ke} (in our experiment $\tau_{ke} = 0.1\%$) leave the VO; entities which maintain this ratio at least at the level of τ_{eff} (in our experiment $\tau_{eff} = 50\%$) form new VOs. VOs with low average effectiveness dissolve and the free-entities join new VOs.

We used 16 processors and we conducted 10 simulation runs for each experiment and present the result. To construct confidence intervals we need much better statistics. Random variables associated with the amount of resources owned by individual producers, the amount of resources per service request, and the service request rate have heavy-tail distributions.

Our results in Figure 3.5(a) show that as the number of service providers increases five fold from $1,000$ to $5,000$ the number of VOs almost doubles from about 225 to about 420, while the average number of resource providers in each VO decreases from about $2,900$ to about $1,700$. When we increase the number of service providers five fold, the effectiveness

of each VO, as measured at the end of the initial self-organization stage, decreases from about 34% to about 19% as shown in Figure 3.5(b). This trend is caused by the diminishing number of resource providers per service provider and is consistent with a reduction of the number of resource providers in each VO. The trend in Figure 3.5(b) is to be expected and shows that the model captures the effect of competition among service providers.



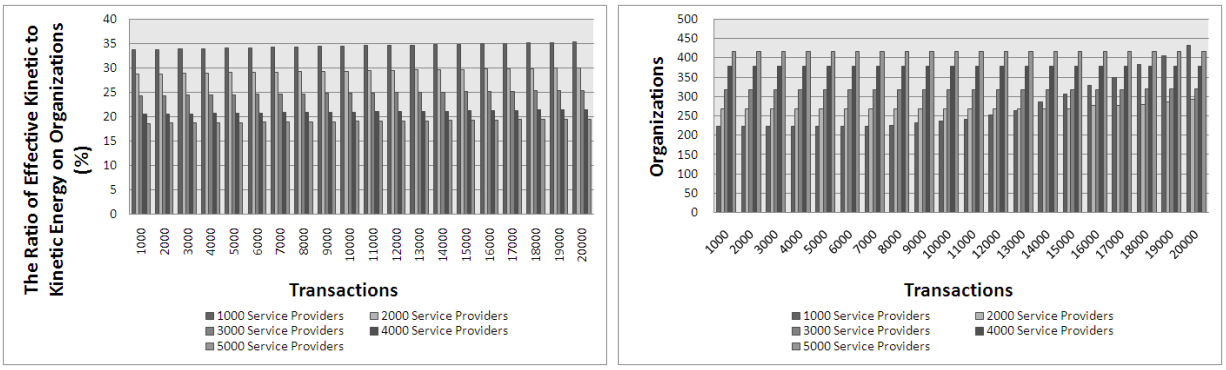
(a) (b)

Figure 3.5: (a)The number of VOs when the number of service providers increases 5 fold from $1,000$ to $5,000$. (b) The average effectiveness of a VO at the end of the initial self-organization phase.

The dynamics of the system is captured by the results showing snapshots of the system after sets of $1,000$ transactions following the initial self-organization phase. In Figure 3.6(a) we see that the effectiveness of the system changes very little from the first snapshot taken after the first $1,000$ transactions till the last snapshot taken after $20,000$ transactions.

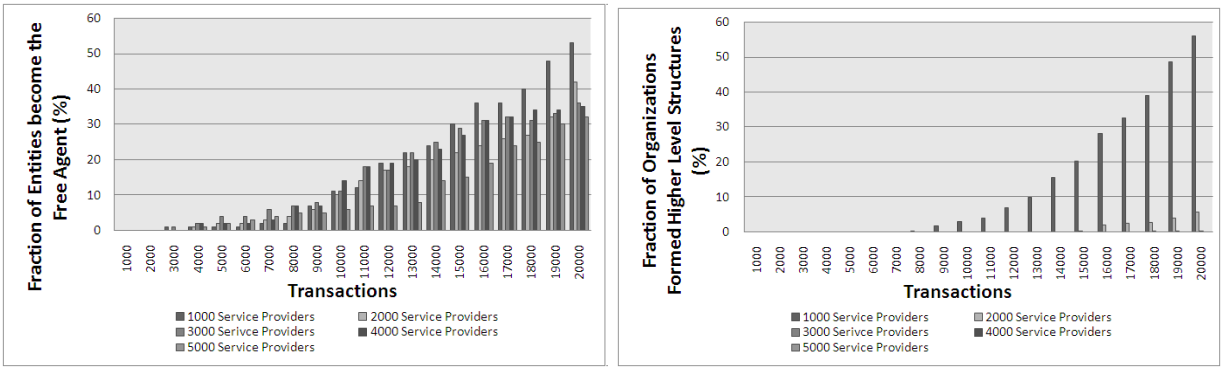
In Figure 3.6(b) we see that as the time progresses, the number of VOs changes from one snapshot to the next and the rate of change increases in some cases. For example, when we have $1,000$ service-providers, we start with some 220 VOs and end with some 420 at the last snapshot after $20,000$ transactions; the increase in the number of VOs is very slight at first and proceeds at a faster pace after $13,000$ transactions. When the number of consumers increases 4 to 5 times we see little change in the number of VOs over time.

Answers to two of the most intriguing questions regarding our model of self-organization are provided by the simulation results in Figures 3.7(a) and (b). The answer to the question



(a) (b)

Figure 3.6: System dynamics. (a) The effectiveness of the system changes little from the first snapshot taken after the first $1,000$ transactions till the last snapshot taken after $20,000$ transactions. (b) The effects of self-organization reflected by the evolution of number of VOs from the first to the last snapshot.



(a) (b)

Figure 3.7: System dynamics. (a) The number of entities which become free agents through 20 consecutive snapshots taken after every 1000 transactions. (b) Creation of new structures, VOs developed within one VO.

if entities which do not meet the efficiency standards become free-agents is answered by the graph in Figure 3.7(a) where we see that as the time progresses an increasingly larger fraction of all entities are forced to join new VOs. The even more interesting questions of whether or not higher level structures are formed by the most performing entities of a VO and whether new structures are formed by the most effective entities in one VO are answered by the graph

in 3.7(b). This optimal behavior is consistent with transitions from one sigmoid to the next as discussed in Section 3.1.

CHAPTER 4: COMMUNICATION SCHEDULING IN SELF-ORGANIZING VERY LARGE SENSOR NETWORKS

Recall from Chapter 1 that: (i) the number of sensors in a VLSN is very large, $N \geq 10^6$; (ii) the sensors are indistinguishable; (iii) the sensors are tiny and inexpensive; (iv) the power reserves and the computing and communication resources are limited; and (v) the sensors are expected to function for a long time in rough conditions.

The self-organization schemes limit the number of partners each sensor collaborates with, thus, it limits the amount of communication and the complexity of coordination. The system is scalable, the amount of state information each node has to maintain is strictly limited regardless of the total number of sensors in the network. The systems we consider mimic biological systems where individual cells of the same type are indistinguishable.

4.1 Sensor Networks

Sensor networks [2, 57, 90] represent a distinct family of wireless networks; they are related to mobile ad hoc networks (MANETs). The nodes of ad hoc and sensors networks form an infrastructure, maintain the network organization in face of mobility, and route messages; the protocols for such networks allow efficient startup and steady state operation.

The organization, routing, and mobility-management of ad hoc networks aim to maximize the throughput and minimize the delay. The objective of the organization, routing, and mobility in a sensor network is to extend the lifetime of the network. Even though technological advances translate into higher processing rates and storage capacity at lower costs, the nodes of future wireless sensor networks will be required to collaborate in order to accomplish any meaningful task. Berkeley Motes and PicoNode, UCLA sensor nodes, and MIT AMPs are some of the devices used in sensor networks. For example, the MICA mote uses an 8-bit microcontroller with 128 kilobytes of flash memory and runs an operating system known as TinyOS [124]. Its radio has a range of a few hundred feet and can transmit

about 40 Kbps; it consumes less than 1 μA when it is off, 10 mA when it receives, and 25 mA when it transmits.

Mobile devices rely on different technologies to communicate depending on the application domain, the transmission range, and several other factors. For example, the Bluetooth technology is widely used for personal communication at a short distance; the nodes communicate using a centrally-assigned time-division multiplexing (TDMA) scheduling strategy with a master node at the center of a star topology and slave nodes synchronized to the master.

Energy efficiency is a major concern for mobile devices as they have limited power reserves and are able to recharge only after extended periods of time [21, 25, 39, 46, 72, 93, 94, 110]. Communication among the mobile nodes is more energy-intensive than either computing or sensing; the energy to transmit 1 kB of data at a distance of 100 m is equal to the energy required by a processor using the year 2000 solid state technology to execute 10^7 instructions [99]. The laws of physics limit the ability of new communication and energy storage technologies to match the sharp reduction of energy consumption per instruction we have witnessed throughout the last decades; we expect that communication efficiency will continue to be an important design goal for sensor networks. This explains why the research related to sensor networks is focused on energy efficient physical and MAC layer protocols [93,112,122,126], routing and topology maintenance [21,39,46,58,110,127], coordination, synchronization, and information dissemination [25], information assurance [49, 86], and reliability [123].

In this section we focus on self-organization and do not address either security, or synchronization. Some of the security aspects of the self-organizing VLSNs are outlined in [74]. Here we only observe that the carrier frequency and the time when the sensors transmit are random, determined by the seeds known only to the sensors in a batch; it is hard, but not impossible, for an intruder to monitor for a long time transmissions over a wide range of

carrier frequencies to guess the pattern of frequency and time of transmission and then to jam.

4.2 Assumptions

The scheme we propose is based upon the following assumptions:

(i) The inexpensive sensors have "genetic information" including a random number generator, seeds, and other network parameters. During the fabrication process the seeds are randomly chosen and the set of seeds is burned-in the ready-only memory of all sensors in the batch. The sensors are tamper-proof, thus it is unlikely to learn the genetic information by disassembling a sensor. The seeds $\alpha_q, 1 \leq q \leq 3$ are used as follows: α_1 to determine a random sequence of events occurring at time slots t_i; α_2 to determine a random sequence of carrier frequencies λ_i; and α_3 to determine a random hopping frequency $\xi_j^{\lambda_i}$ for each carrier frequency λ_i. The genetic material also includes: κ, the number of events in one phase and ν, the number of activity phases, as well as μ, the cardinality of the proximity set, M, the cardinality of the event index list used to keep track of transmission and receiving events, and φ, a parameter for synchronization.

(ii) The average transmission range of a sensor is γ and we expect to have on average ρ sensors per unit of the area covered by the network.

(iii) The network is dense; this means that $\zeta = \pi \times \gamma^2 \times \rho$, the average number of sensors that are able to receive the transmission of a sensor is at least $\zeta \geq p \times \mu$ with p a small integer, $4 \leq p \leq 6$ and μ the cardinality of the proximity sets.

(iv) The sink Σ has a larger power reserve and transmission range. It links the sensor network with the outside world and communicates with an external controller (a satellite, a drone, or even a stationary device) to report relevant information. For now we assume a unique sink, but fault-tolerance requires backup sinks that can take over if the original sink fails. Initially, all sensors are synchronized to the sink.

(v) The sensors are *reactive* in terms of communication. A sensor responds to a successful transmission of another sensor after evaluating two *fitness functions* f and g; only if the value of the fitness function exceeds a certain threshold the sensor is allowed to transmit. This threshold depends on several parameters including the strength of the incoming signal and the power reserve of the sensor. The fitness function and the determination of the threshold are fairly complex subjects [110] and are not discussed in this dissertation.

(vi) A sensor dwells φ micro-seconds on each frequency in hopping sequence. When a sensor wakes up at the time of the event ϵ_i, its master clock is in one of the time slots t_{i-1}, t_i, or t_{i+1}. Each sensor knows the frequencies λ_{i-1}, λ_i, and λ_{i+1} on which the sender will dwell in the time slots t_{i-1}, t_i, and t_{i+1}. A strategy to allow synchronization in the presence of clock drift is: tune in, cyclically, to λ_{i-1}, λ_i, and λ_{i+1} spending $\varphi/3$ time units on each of them.

4.3 Events and Epochs

The time evolution of the network consists of several *epochs*, each one starting with a *self-organization (set-up)* phase followed by a number ν of *activity (steady-state)* phases, Figure 4.1; the names of the phases are suggestive and an in-depth discussion of the role and the function of each phase is deferred for later sections. We use the term *event* to describe a communication event, the transmission of a message at the beginning of a time slot; the index k of the event ϵ_k occurring at time t_k, the beginning at slot k, reflects the order of the event: if $k_1 \leq k_2$ then $\epsilon_{k_1} \mapsto \epsilon_{k_2}$ (ϵ_{k_1} before ϵ_{k_2}). All phases consist of the same number of events, κ. We can refer to an event by its global index, its index within an epoch, and its index within a phase, Figure 4.2. The index k of ϵ_k is a global pointer into a random sequence of frequencies and time slots. The *reciprocal* of event ϵ_k during the self-organization phase is the event $\epsilon_{\bar{k}}$ during an activity phase; the reciprocal index is $\bar{k} = \kappa - k - 1$. Reciprocal events occur in reverse order, if $k_1 \leq k_2$ then $\bar{k}_2 \leq \bar{k}_1$ and $\epsilon_{\bar{k}_2} \mapsto \epsilon_{\bar{k}_1}$.

Table 4.1: Summary of Notations

N	number of sensors in the batch
$\mathcal{P}(sigma_i)$	proximity set of sensor σ_i
μ and μ_{actual}	maximal and actual cardinality of the proximity set
κ	number of events in a phase
ν	number of activity phases
η	number of events in one epoch
M	maximum number of events in *EventList*
γ	transmission range
ρ	average sensor density
θ	standard deviation of sensor density
ϵ_k	k-th event during the self-organization phase
$\epsilon_{\bar{k}}$	reciprocal event of ϵ_k during an activity phase
φ	parameter for synchronization
ζ	average number of sensors receiving a transmission
λ_k	carrier frequency in slot k
$\xi_j^{\lambda_k}$	hopping frequency around λ_k
E_o^t and E_o^r	energy for transmission and reception during a self-organization phase
E_a^t and E_a^r	energy for transmission and reception during an activity phase
ω	expected number of collisions in a CRI
δ	the duration of a micro-slot
$\Delta = n_\delta \delta$	the duration of a slot ($n_\delta = \mathcal{O}(10^3)$)

The ordering of events is $1 \mapsto 2 \mapsto 3 \ \ldots \ \mapsto \kappa - 1 \mapsto \kappa$ during the self-organization phase and $\bar{\kappa} \mapsto \overline{\kappa - 1} \mapsto \ \ldots \ \mapsto \bar{3} \mapsto \bar{2} \mapsto \bar{1}$ during an activity phase.

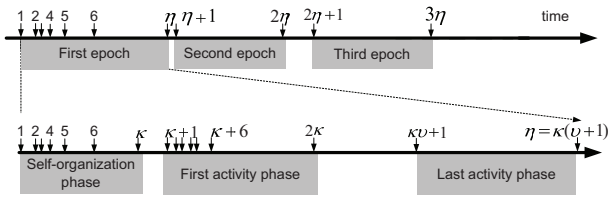


Figure 4.1: Epochs, phases, and events. The number of events in an epoch is η and the number of events in a phase is κ. Each epoch consists of one self-organization phase followed by ν activity pases. Thus, $\eta = \kappa(\nu + 1)$.

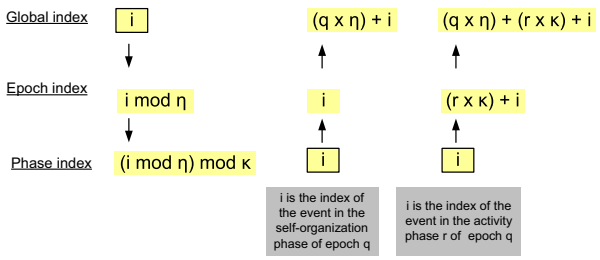


Figure 4.2: Global, epoch, and phase indices of an event. If i is the global index then the epoch index is $(i \ mod \ \eta)$ and the phase index is $[(i \ mod \ \eta) \ mod \ \kappa]$. If i is the index of an event in the self-organization phase of epoch q, then its phase index is i, and the global index is $(q \times \eta + i)$. If i is the index of an event in the r-th activity phase of epoch q then its phase index is $(r \times \kappa + i)$, and the global index is $(q \times \eta + r \times \kappa + i)$.

During the self-organization phase a sensor σ_i assumes a globally unique pseudo-identity, $Pid(\sigma_i)$. Initially the Pid of each sensor is set to zero. $Pid(\sigma_i) = k$, if during the self-organization phase σ_i successfully transmits during slot $(k-1)$, possibly after a set of collisions, and wins the right to be the only sensor allowed to transmit at the beginning of slot k. The Pid is included in every message sent by σ_i during the self-organization phase to ensure that the proximity set does not grow beyond the limit imposed, $\mid \mathcal{P}(\sigma_i) \mid \leq \mu$ and also that σ_i does not appear multiple times in any proximity set $\mathcal{P}(\sigma_j)$. The Pid is available during the activity phase, but not used for now.

The expected duration of an epoch is application dependent and it is controlled by η, the number of events per epoch. Some of the factors that affect the choice of η are: the duration of the deployment, the expected life-time of individual sensors, the intensity of communication during the deployment, and the frequency of topological changes. An epoch could last minutes for an intense and dynamic application when new sensors are added frequently and the life-time of sensors is very limited because they deplete their power at a high rate, e.g., monitoring and control of a forest-fire. An epoch could be of the order of days or even months for a low-intensity application with a relatively stable topology, e.g., long term monitoring of volcanic activity. The expected duration of an epoch can be controlled

by scaling the random numbers dictating the timing of events, e.g., using seconds, minutes, hours, and so on, as units of time.

Communication efficiency is a critical aspect of sensor networks and in the next section we discuss our proposal for a MAC layer algorithm which allows individual sensors to acquire a unique pseudo-id and as we shall see later to establish collision-free communication channels and a schedule when they need to wake up.

4.4 Integrated Medium Access Control (MAC) and Self-organization Algorithms

There are two basic classes of strategies for sharing a communication channel: scheduled and non-scheduled multiple access. Both strategies are represented among the Medium Access Control (MAC) layer protocols for ah hoc and sensor networks. Among the strategies based upon scheduled access we mention: Code Division Multiplexing (CDMA) which employs spread-spectrum technology and a special coding scheme (where each transmitter is assigned a code) to allow multiple users to be multiplexed over the same physical channel; Time-Division Multiple Access (TDMA) divides access by time, while Frequency-Division Multiple Access (FDMA) divides it by frequency.

Several MAC-layer protocols for ad hoc networks avoid, or reduce collisions. A non-exhaustive list of such protocols includes: Medium Access Collision Avoidance (MACA) which uses Request to Transmit and Clear to Transmit (RTS/CTS) messages to avoid the *hidden node problem* [55]; Medium Access protocol for Wireless LANs (MACAW), a protocol similar to MACA but using additional ACK (Acknowledgment) and backoff mechanisms [19]; Power Aware Multi Access protocol with Signaling for Ad Hoc Networks (PAMAS) [104] which use one channel for control packets and one for data packets; Carrier Sense Medium Access with Collision Avoidance (CSMA/CA) [34]; and IEEE 802.11 [51] which takes advantage of RTS/CTS/DATA/ACK packets and physical and virtual carrier sensing for collision avoidance.

The SMACS (Self-organizing Medium Access Control for Sensor Networks) protocol [99] uses a TDMA-like frame combined with FDMA CDMA to avoid interference among nodes. The *organized* channel access method used by SMACS and by the protocol in this dissertation can be traced back to several papers [7, 41] which propose to form a hierarchical structure to localize groups of nodes and make the channel assignment easier.

Several other TDMA-based MAC protocols for sensor networks have been proposed including WLC12-5 [88]. Sensor MAC (S-MAC) is a protocol inspired by PAMAS and implemented over Berkeley Motes [126]. The nodes listen and sleep periodically; the radio is set to sleep during transmissions of other nodes. Neighboring nodes form virtual clusters to auto-synchronize on sleep schedules. The protocol divides long messages into small segments and transmits all segments back to back; it uses RTS/CTS once per message but ACK for each segment. The energy savings are 2.5 times larger than for IEEE 802.11.

To increase the algorithm efficiency the number of collisions experienced by a sensor when it transmits should be minimized and a sensor should be idle as long as feasible. Thus, it is highly desirable to integrate the MAC protocol/algorithm with the algorithm for self-organization which determines the schedule for communication. This requirement precludes the use of one of the existing MAC algorithms; we decided to adapt a known Collision Resolution Algorithm (CRA) and integrate it with our self-organization scheme.

A fair number of Collision Resolution Algorithms for Random Multiple-Access are extensively analyzed in the literature [38]. The basic idea of the algorithms is to split recursively the set of nodes involved in a collision on a multiple-access channel until the cardinality of the set is equal to one and a single node successfully transmits. This splitting is based upon the ternary channel feedback, "Success," "Collision," or "Idle Slot". *Blocking* algorithms forbid new nodes to join the set of nodes involved in a collision; newcomers have to wait until the original collision was resolved and only then transmit. *Non-blocking* algorithms allow newcomers to join a game in progress. In this dissertation we consider the so called

Stack Algorithm also known as CTM (Capetanakis-Tsybakov-Mihailov) [107], Figure 4.3 (a); CTM is non-blocking.

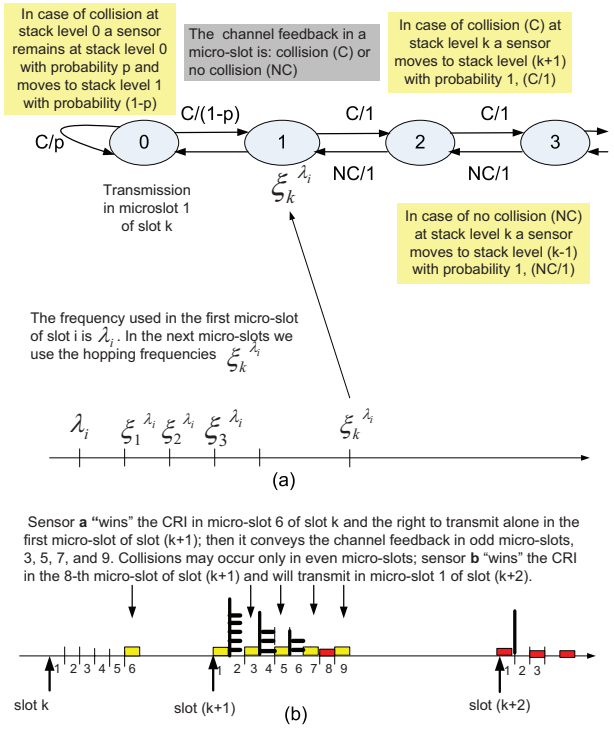


Figure 4.3: (a) The stack collision resolution algorithm. Each sensor creates a virtual stack and once involved in a collision updates the stack based upon the channel feedback (collision/no-collision). Only sensors at stack level 0 are allowed to transmit during the collision resolution period. (b) Modified stack algorithm. The original sender conveys the channel feedback. In this case, sensor a transmits successfully in the 6-th micro-slot of slot k and invites other sensors to transmit. Multiple sensors including b respond to the invitation; collisions involving sensor b occur in the micro-slots $2, 4$ and 6 of slot $(k+1)$; finally, in micro-slot 8 sensor b is the only one allowed by the algorithm to transmit. Sensor a conveys the channel feedback in micro-slots $3, 5, 7$, and in micro-slot 9 of slot $(k+1)$ it announces that sensor b was successful and has won the right to transmit undisturbed in the first micro-slot of slot $(k+2)$.

The model of the multi-access channel suitable for a sensor network is slightly more complicated than the one considered by traditional collision resolution algorithms. First, the transmission frequency changes, and, most importantly, not all sensors involved in a collision are able to hear the channel feedback. The transmission range of the sensors is normally distributed around γ. It is thus possible that two sensors b and c both transmit in

response to a request from sensor a, but they are outside of the reception range of each other, a phenomenon encountered in wireless networks and called *the hidden node problem* [110]. The version of the stack algorithm used in this dissertation, Figure 4.3 (b), differs from the original CTM algorithm in several ways:

(i) The algorithm is executed only during the self-organization phase when collisions may occur.

(ii) During the self-organization phase the duration of a slot is $\Delta = n_\delta \delta$ with δ the duration of a micro-slot and $n_\delta = \mathcal{O}(10^3)$. The time t_k of the event ϵ_k is the starting time of slot k and of its first micro-slot. The time between two consecutive events is determined by the random number generator, and should be at least Δ, $(t_{k+1} - t_k \geq \Delta)$. We expect a collision in slot k to be resolved well before the event ϵ_{k+1} as $n_\delta = \mathcal{O}(10^3)$.

(iii) Only one sensor is allowed to transmit in the first micro-slot of a slot: the *sink* transmits during the first slot; the "winner" of a collision resolution contest during slot $(k-1)$, transmits in slot k.

(iv) The micro-slots are grouped in pairs. Collisions may occur only in the even micro-slots of a slot. Odd micro-slots are collision-free and used by the sensor which initiated the CRI to broadcast the channel feedback. This allows us to address the hidden node problem. Indeed two sensors b and c may be in the range of a and may attempt to responding to a in micro-slot $2k$, but they may be out of each other's range. To solve this problem a will broadcast in micro-slot $2k + 1$ the channel feedback and in this case report a collision.

(v) The algorithm is blocking, only the sensors involved in the initial collision are allowed to compete.

(vi) Each sensor involved in a collision maintains a counter of the micro-slots involved to determine the hopping frequency used to transmit in any micro-slot.

The algorithm is distributed in time and space; each sensor involved in a transmission maintains a virtual stack and updates it according to the channel feedback, Figure 4.3(a).

The splitting algorithm guarantees that only one sensor can respond successfully in a Collision Resolution Interval (CRI) thus, the *Pid* of a sensor is unique. Only sensors at stack level 0 are allowed to transmit in any even micro-slot; initially, all sensors wishing to transmit in the second micro-slot of the slot k use frequency λ_k and set their stack level to 0. If there is a successful transmission (this happens when only one sensor transmits) then the CRI terminates after the sensor which transmitted in the first micro-slot announces the winner. If there is no transmission, one of the sensors sends a *ReqToForward* message in response to an "Idle Slot" channel feedback using the hopping frequency $\xi_1^{\lambda_k}$. When multiple sensors transmit in a micro-slot a collision occurs; then all sensors update their stack as follows: remain at stack level 0 with probability p_s and move to stack level 1 with probability $(1 - p_s)$. It is likely that the cardinality of the set of sensors at stack level 0, allowed to transmit during the next pair of micro-slots will be smaller; these sensors use the first hopping frequency $\xi_1^{\lambda_i}$. The process continues; if a new collision occurs in the next pair of micro-slots all sensors at stack level zero repeat the splitting process, while those at stack level 1 move to stack level 2 with probability one.

4.5 Pseudo-id, Proximity and Reverse Proximity Set

During the self-organization phase a sensor assumes a globally unique pseudo-identity (*Pid*) given by the index of the first event when it has transmitted successfully. Before the first successful transmission, the *Pid* of each sensor is set to zero and then reset to the index of the slot when the successful transmission takes place. The *Pid* is stamped on every message sent during the self-organization phase to guarantee that the proximity set of a sensor does not grow beyond the limit imposed.

The *proximity set* of sensor A is a subset of neighbors within the range of A which receive information from the sensor A during an activity phase. *The reverse proximity set* is a subset of neighbors within the range of sensor A. If B is in the reverse proximity set of A then A (as

well as the other sensors in the proximity set of B) is required to acquire during an activity phase. Each sensor maintains information about its reverse proximity set: a list of all sensors which have accepted it as a member of their proximity sets and a list of events when it is expected to wake up and receive a transmission from each one and then to transmit at the next event. The sensors do not maintain information about their own proximity set.

The information maintained by a sensor is summarized in Figure 4.4.

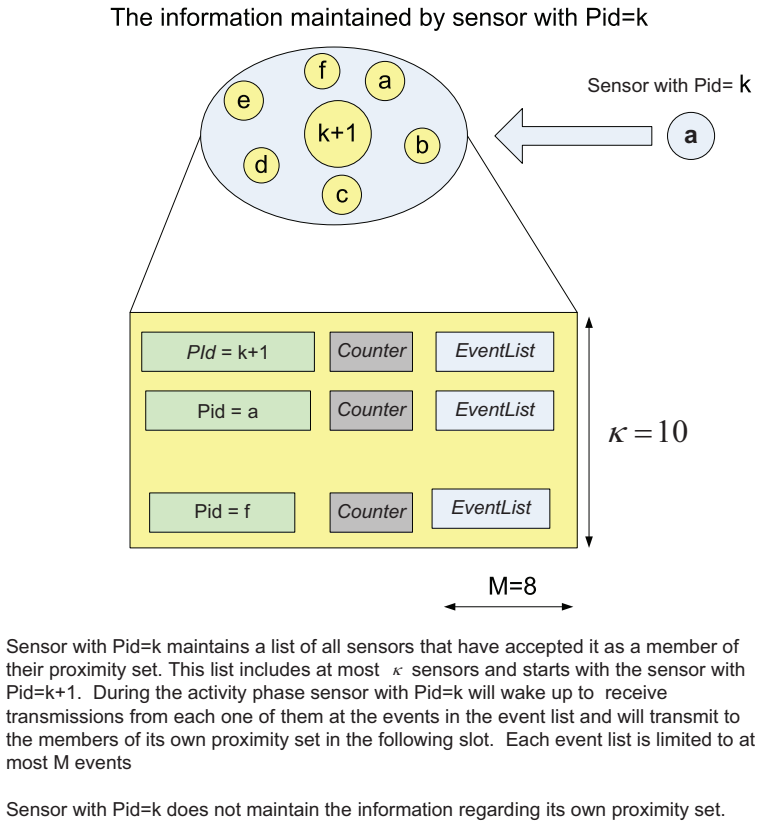


Figure 4.4: The information maintained by a sensor relates to its reverse proximity set.

Recall that the scheme we propose covers epochs of η events; an epoch starts with a self-organization phase followed by multiple activity phases. Collisions occur only during the self-organization phase when the network is organized based upon physical attributes such as proximity, residual power, type of information the sensor is collecting, and possibly other attributes.

Next we discuss the self-organization phase.

The role of the self-organization phase is to establish the communication pattern between a sensor and the members of its proximity set during all activity phases following a self-organization phase. At the conclusion of the self-organization phase the node with $Pid = k$ knows the index of the events when it is scheduled to wake-up to receive a message from one of the sensors which accepted the sensor in its proximity set and then transmit to all members of its own proximity set. Knowing the index k of the event ϵ_k does not mean that we know the time t_k of occurrence of the event. The times t_k are random, but all sensors can determine them, as they share the seeds for the random number generators used to calculate the times t_k.

4.6 Case Study: SFSN - A Scale-free Self-organizing VLSN

The self-organization strategy supported by the SFSN algorithm guarantees that every node is connected with a limited number of nodes. This means that a sensor $\sigma_i, 1 \leq i \leq N$ is able to construct a *proximity set*, $\mathcal{P}(\sigma_i)$, of neighboring sensors it communicates with. The network is scale-free, $\mid \mathcal{P}(\sigma_i) \mid \leq \mu$, regardless of the number N of sensors of the network; σ_i maintains a limited amount of state information regarding the sensors in $\mathcal{P}(\sigma_i)$. For example, the proximity set of sensor a, $\mathcal{P}(a)$, in Figure 4.4 consists of a subset of sensors in the range of a; a must also be in the range of the sensors in $\mathcal{P}(a)$. The quantities μ, M and η, ν, and κ in Figures 4.1 and 4.4 are constants, selected at the time the network is planned; see Table 4.1 for a summary of notations.

An informal high-level description of the self-organization algorithm follows:

(i) A specially configured node called the *sink*, initiates the self-organization process at the time t_1 of the first event ϵ_1 and requests to be included in the proximity set of one of the nodes in its vicinity and assumes a *Pid* of 1.

(ii) Multiple sensors receive the request and decide whether to respond or not; if more than one responds, a Collision Resolution Algorithm (CRA) is used and eventually sensor σ_i transmits successfully in the slot triggered by event ϵ_1 and wins the right to transmit at time t_2 of the second event, ϵ_2. Then σ_i assumes a *Pid* of 2, includes the *sink* in its proximity set, $\mathcal{P}(2)$, and records that during the activity phases must wake-up at reciprocal event $\bar{1}$ to receive a transmission from the *sink* and then transmit at reciprocal event $\bar{2}$ to a member of its own proximity set. At the time of the second event ϵ_2, the role of the *sink* is played by σ_i which sends a request to join the proximity set of one of its neighbors. Eventually, a sensor σ_j accepts; then σ_j assumes a *Pid* of 3 and includes σ_i in its proximity set, $\mathcal{P}(3)$. Later, when σ_j receives another request from $\sigma_i \in \mathcal{P}(3)$ it records only the index of the event in the *EventList* corresponding to *Pid*=2.

(iii) The process continues and eventually the *sink* responds to a request from a sensor σ_k and includes σ_k in $\mathcal{P}(1)$. Recall that the *sink* has a *Pid* of 1 and each sensor assumes as *Pid* the index of the first event when it sends a request to join a proximity set.

(iv) The self-organization phase ends after κ events and an activity phase starts. During the activity phase σ_i wakes up at the time of the events in its *EventList* to receive from one of the members of its proximity set and then transmit in the next slot to σ_j such that $\sigma_i \in \mathcal{P}(\sigma_j)$. At the end of each activity phase the sink reports to an external monitor.

Three types of messages: *ReqToJoin* (RtJ), *ReqToForward* (RtF), and *AcceptToJoin* (AtJ) are exchanged during the self-organization phase. *ReqToJoin* expresses the desire of the sender to join the proximity set of one of its neighbors; *ReqToForward* signals that no neighbor is willing to accept the sender to join its proximity set and one of them forwards the message to others to keep the organization process going. A *ReqToForward* always contains the *Pid* of the originator of a *ReqToJoin* message, rather than the *Pid* of the sensor forwarding the message. An *AcceptToJoin* is sent in response to a *ReqToJoin*. Sensor σ_j uses two fitness functions to determine how to respond to a successful transmission of a

ReqToJoin from σ_i: f_j to determine whether it should respond with an *AcceptToJoin* and g_j to determine whether it should respond with a *ReqToForward*.

When σ_i includes in its proximity set a sensor with *Pid*=$\mathcal{I}_{sender}$ it also creates a counter of events when it has received messages from it, $Count(\mathcal{I}_{sender})$, as well as an event index list $EventList(\mathcal{I}_{sender})$, of size at most M, Figure 4.4. It follows that the storage requirements for control information for each sensor are: $\mu(2+M)$ integers, Figure 4.4. The self-organization algorithm followed by sensor σ_i with proximity set $\mathcal{P}(\sigma_i)$ and fitness functions f_i and g_i when σ_i observes a successful transmission in the slot of ϵ_k from $\mathcal{I}_{sender}$ is shown in Algorithm 8.

Occasionally, a sensor receiving a *ReqToJoin* does not accept the sender as a member of its proximity set. In the example in Figure 4.5, b responds to a *ReqToJoin* from a with an *AcceptToJoin*; a joins $\mathcal{P}_b$ in slot k, then b sends its own *ReqToJoin* in the first micro-slot of slot $(k+1)$. After noticing an empty micro-slot in slot $(k+1)$ sensor c which has its proximity set full, sends a *ReqToForward* which reaches sensor d; then d sends an *AcceptToJoin* and b joins $\mathcal{P}_d$. Next d sends a *ReqToJoin* in slot $(k+2)$ answered by e and the process continues, d joins $\mathcal{P}_e$ and so on. This strategy allows the self-organization to continue even when sensors have their proximity sets full.

During an activity phase the sensors carry out their monitoring function and report partial results. The schedule of events for each sensor, namely the slots when it transmitted successfully during the self-organization phase is known, and this knowledge allows each sensor to determine the slots during the activity phase when they have to wake-up to receive information and then transmit.

The event $\epsilon_{\overline{k}}$ in an activity phase is the reciprocal of the event ϵ_k in the self-organization phase; if sensor σ_i transmitted in the first micro-slot of slot k during the self-organization phase then it will be scheduled to wake-up and receive in slot $\overline{k-1}$, where $\overline{k-1} = \kappa - k$, and then transmit in slot $\overline{k}$. During the activity phases sensors f, e, d, c, b and a in Figure

```
if f_i returns True when receiving a message in one of the micro-slots of event k then
    if I_sender ∈ P(σ_i) then
        if Count(I_sender) ≥ M then
            if (g_i return True) and (no transmission at the first micro-slot of event
            k+1) ;
            then
                broadcast RtF message at the second micro-slot of event k + 1;
            end
        else
            if messageType is RtJ then
                Count(I_sender) = Count(I_sender) + 1;
                insert k in EventList(I_sender);
            end
            transmit an AtJ message in the next micro-slot of slot k and follow the
            CRA;
            if transmission is successful then
                add I_sender to P(σ_i), increment Count(I_sender), insert k in
                EventList(I_sender);
                broadcast RtJ message at the first micro-slot of event k + 1;
            end
        end
    else if I_sender ∉ P(σ_i) then
        if |P(σ_i)| ≥ μ then
            if (g_i return True) and (no transmission at the first micro-slot of event
            k+1) then
                broadcast RtF message at the second micro-slot of event k+1;
            end
        else
            transmit an AtJ message in the next micro-slot of event k and follow the
            CRA;
            if transmission is successful then
                add I_sender to P(σ_i), increment Count(I_sender), insert k in
                EventList(I_sender);
                broadcast RtJ message at the first micro-slot of event k + 1;
            end
        end
    end
else if f_i returns False when receiving an incoming message then
    if (g_i return True) and (no transmission at the first micro-slot of event k + 1)
    then
        broadcast RtF message at the second micro-slot of event k+1;
    end
end
```

Algorithm 8: The self-organization algorithm followed by sensor σ_i with proximity set $\mathcal{P}(\sigma_i)$.

4.5 transmit in slots $\overline{k+5}, \overline{k+4}, \overline{k+3}, \overline{k+2}, \overline{k+1}$ and $\overline{k}$ respectively and the members of their respective proximity sets wake-up.

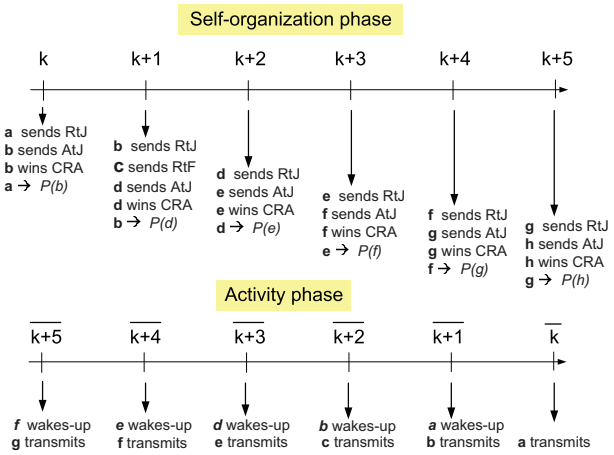


Figure 4.5: Handling of *ReqToForward* during the self-organization and activity phases.

The algorithm followed by sensor σ_i in an activity phase is:

(i) Compute the index of reciprocal events in the *EventList* for all members of $\mathcal{P}_i$.

(ii) Construct an ordered list of reciprocal events.

(iii) Compute the time of the next event in the ordered list; wake up and receive at that time. Compute the time of the next event and transmit at that time. Increment the count of events.

(iv) Repeat the previous step until all reciprocal events have been exhausted.

In the example in Figure 4.5 during the self-organization phase the sensors a, b, c, d, e, f, and g transmit in slots $k, k+1, k+2, k+3, k+4$, and $k+5$, respectively. During the activity phases sensors f, e, d, c, b, and a transmit in slots $\overline{k+5} = \kappa - k + 4, \overline{k+4} = \kappa - k + 3, \overline{k+3} = \kappa - k + 2, \overline{k+2} = \kappa - k + 1, \overline{k+1} = \kappa - k$, and $\overline{k} = \kappa - k - 1$, respectively, and the members of their respective proximity sets wake-up.

SWAS, the algorithm introduced in this dissertation shares some ideas with the SAS algorithm [78] and the Self-organizing Medium Access Control for Sensor Networks (SMACS) [99]. SWAS, SAS and SMACS algorithms first determine the radio connectivity in the network and then assign collision-free channels to the links; all three assume limited mobility. Unlike the Link Clustering Algorithm in [7] which performs two passes, the first carried over the entire network to discover neighbors and the second to assign channels to links between two neighbors, SWAS, SAS and SMACS algorithms assign immediately a channel to a link. Other similarities: all algorithms assume that nodes are able to turn their radio on and off; the nodes are able to tune the carrier frequency to different bands and the number of available bands is quite large.

The major differences between SWAS and SAS algorithms, on one side, and the SMACS, on the other side are: (i) The nodes of SWAS and SAS networks are anonymous, they do not have a physical address and they communicates using multiple unidirectional channels; (ii) For SWAS and SAS the reorganization of the network occurs periodically, while for SMACS there is only one setup phase followed by a steady-state operation mode; (iii) The virtual channels assigned during the self-organization phase in SWAS and SAS networks are implicit and related only to the index of the communication event, the nodes do not exchange a schedule for communication. In SMACS algorithms the assignation of the channels is explicit, the carrier frequencies are chosen once and for all and two nodes exchange the schedule of transmissions for the entire duration of the steady-state operation; (iv) Multiple nodes may choose to respond to an invitation to transmit during the self-organization phase in SWAS and SAS and collisions are likely; though collisions may occur during the assignation of the channel in SMACS, these are rare events; (v) For SWAS and SAS the self-organization phase proceeds strictly sequentially, thus, the setup phase may take longer. The network setup can

be done in parallel in SMACS, multiple nodes may initiate the assignment of the channel at the same time.

The networks topology distinguishes SWAS from SAS algorithms, a Watts-Strogatz graph for the former and a random graph for the later. In both cases the self-organization algorithms and the Medium Access Control (MAC) algorithm are integrated, but they differ substantially. The self-organization strategy of SWAS networks guarantees that every node is connected with a limited number of nodes. A sensor $\sigma_i, 1 \leq i \leq N$ is able to construct a *proximity set*, $\mathcal{P}_i$, of neighbors it communicates with. The network is scale-free, $\pi_i = \mid \mathcal{P}_i \mid$ is limited regardless of the number N of sensors in the batch; σ_i maintains a limited amount of state information regarding the sensors in $\mathcal{P}_i$.

The SWAS extends the SFSN algorithm by creating the shortcuts between two remote nodes in order to reduce the average path length and increasing the level of clustering. We present an informal description of the SWAS algorithm below.

The term *event* means a communication event, the transmission of a message at the beginning of a time slot; the event ϵ_k occurs at time t_k, and marks the beginning of slot k. If $k_1 \leq k_2$ then $\epsilon_{k_1} \mapsto \epsilon_{k_2}$ (ϵ_{k_1} before ϵ_{k_2}). The index k of ϵ_k is a global pointer into a random sequence of time slots and carrier frequencies. The evolution in time of the system consists of several *epochs*, each one starting with a *self-organization (set-up)* phase followed by a number ν of *activity (steady-state)* phases, each one with κ events. The self-organization phase consists of a *regular topology build-up* period with κ_r events leading to the formation of a logical ring, including the sink, followed by a *small-worlds topology build-up* period with κ_{sw} events and $\kappa_r + \kappa_{sw} = \kappa$.

We can refer to an event by its global index, its index within an epoch, and its index within a phase. If i is the global index then the epoch index is (i *mod* η) and phase index is [(i *mod* η) *mod* κ]. If i is the index of an event in the self-organization phase of epoch q, then its phase index is i, and the global index is $(q \times \eta + i)$. If i is the index of an event in the r-th activity phase of epoch q then its phase index is $(r \times \kappa + i)$, and the global index is

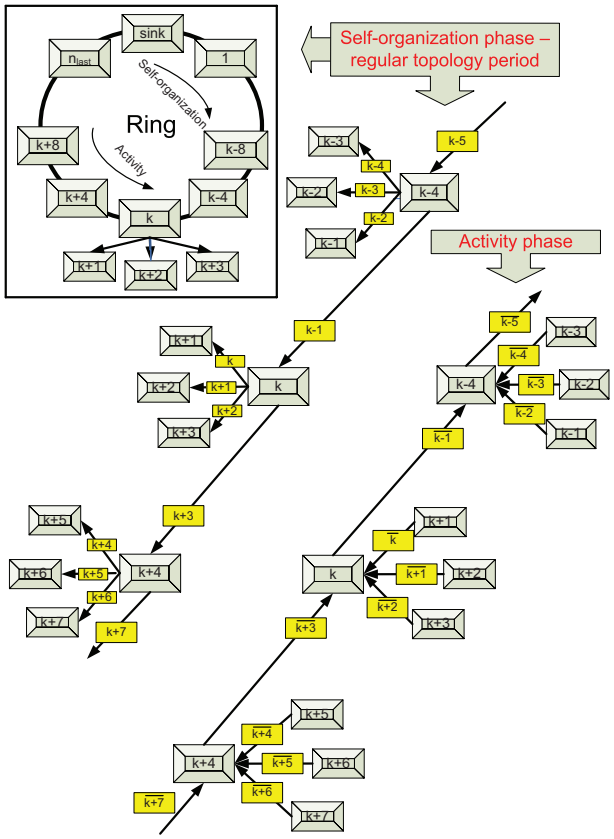


Figure 4.6: Self-organization and activity phases. The creation of the regular topology - a ring during the first sub-phase of self-organizations when $\mu = 3$. The sensor with $Pid = k$ accepts sensor with $Pid{=}k-4$ to join its proximity set, $\mathcal{P}_k$ at the time of the event ϵ_{k-1}. Then sensor k requests and it is accepted to join the proximity sets of sensors with $Pid{=}k+1, k+2, k+3$ at the time of the events $\epsilon_k, \epsilon_{k+1}$ and ϵ_{k+2}; finally, at the time of the event ϵ_{k+3}, sensor k requests and is accepted to join the proximity set of sensor $k+4$, its successor in the ring. During the activity phase sensor k receives the communication from the sensor with $Pid{=}k+4$ in slot $\epsilon_{\overline{k+3}}$ and from sensors with $Pid{=}k+3, k+2, k+1$ in slots $\epsilon_{\overline{k+2}}, \epsilon_{\overline{k+1}}$, and $\epsilon_{\bar{k}}$, respectively, and transmits to the sensor with $Pid{=}k-4$ in slot $\epsilon_{\overline{k-1}}$.

$(q \times \eta + r \times \kappa + i)$. The *reciprocal* of event ϵ_k occurring during the self-organization phase is the event $\epsilon_{\bar{k}}$ during an activity phase; the reciprocal index is $\bar{k} = \kappa - k - 1$. Reciprocal events occur in reverse order, if $k_1 \leq k_2$ then $\bar{k}_2 \leq \bar{k}_1$ and $\epsilon_{\bar{k}_2} \mapsto \epsilon_{\bar{k}_1}$. The ordering of events is: $1 \mapsto 2 \mapsto 3 \ \ldots \ \mapsto \kappa - 1 \mapsto \kappa$ and $\bar{\kappa} \mapsto \overline{\kappa - 1} \mapsto \ \ldots \ \mapsto \bar{3} \mapsto \bar{2} \mapsto \bar{1}$ during the self-organization and an activity phase, respectively.

A sensor can be a cluster head and be connected to the ring, or a member of a cluster, thus, connected to a cluster head which in turn is connected to the ring. A cluster head is a member of the proximity sets of all the sensors in its cluster and of its successor in the ring.

During the self-organization phase a sensor assumes a globally unique pseudo-identity *Pid*. The *Pid* is included in every message sent by a sensor during the self-organization phase and it is available during the activity phase, but not used for now. Initially, the *Pid* of every sensor other than the *sink*, a special sensor with *Pid*=1, is set to zero. The *Pid* of a sensor is set to the index of the slot during the first period of self-organization following the slot when the sensor has for the first time accepted another sensor in its proximity set. If $Pid = k$ this means that the sensor has transmitted successfully in slot $(k-1)$, possibly after a set of collisions, and accepted a sensor with $Pid = k-1$ in its proximity set. The *Pid*s of cluster headers are $k = 1 + (\mu + 1), \ldots, 1 + (i \times \mu + 1), \ldots$ A cluster head with $Pid = k$ wins the right to transmit at the beginning of the slots $k, k+1, \ldots k + (\mu - 1)$ without any interference from any other sensor. The sensors which invite sensor k to join their proximity sets at the time of the events $\epsilon_k, \epsilon_{k+1}, \ldots, \epsilon_{k+\mu-1}$ assume $Pid = k+1, k+2, \ldots, k+\mu$ and become members of its cluster; these sensors do not attempt to join the proximity set of other sensor.

The self-organization phase consists of two periods, the first to form of a clusters and then a ring of cluster heads and the second to create shortcuts leading to a small-worlds topology. An informal high-level description of the self-organization algorithm follows: (i) A specially configured node called the *sink* with *Pid*=1, initiates the self-organization process at the time t_1 of the first event ϵ_1, by requesting to be included in the proximity set of one of the nodes in its vicinity. (ii) Multiple sensors receive the request and decide whether to respond or not; if more than one responds, a Media Access Control (MAC) is used and eventually sensor σ_i transmits successfully in the slot triggered by event ϵ_1 and, wins the right to transmit at time $t_2, t_3, \ldots, t_{\mu+1}$ of events $\epsilon_2, \epsilon_3, \ldots \epsilon_{\mu+1}$. Then σ_i assumes a *Pid*=2, includes the *sink* in its proximity set, $\mathcal{P}_2$. At the time of the events $\epsilon_2, \epsilon_3, \ldots, \epsilon_{\mu+1}$, sensor σ_i

known from now on by Pid=2, sends requests to join the proximity sets of μ of its neighbors. In each of these slots eventually, sensors $\sigma_{j_1}, \sigma_{j_2}, \ldots \sigma_{j_\mu}$ accept the invitation, assume Pid $3, 4, \ldots 2+\mu$ and include sensor with Pid=2in their proximity sets, $\mathcal{P}_3, \mathcal{P}_4, \ldots \mathcal{P}_{2+\mu}$. The sensors $\sigma_{j_1}, \sigma_{j_2}, \ldots \sigma_{j_\mu}$ form a cluster with the sensor Pid=2 as cluster head. They monitor the formation of the regular network but do not attempt to join the proximity set of any sensor. Then the sensor with $Pid = 2$ is included in the proximity set of a sensor which assumes the $Pid = 3+\mu$ and becomes the next neighbor in the ring as shown in Figure 4.6. Lastly, the sensor with $Pid = 2$ records that during the activity phases must wake-up at reciprocal events $\epsilon_{\overline{\mu+2}}, \epsilon_{\overline{\mu+1}}, \ldots, \epsilon_{\bar{2}}$ to receive a transmissions from the sensors whose proximity set it joined during the self-organization phase and then transmit to the *sink* at $\epsilon_{\bar{1}}$ as seen in Figure 4.6. Then the sensor with Pid= $3+\mu$ repeats the process and the process of construction of the ring continues. (iv) Eventually a sensor with Pid close to N, the number of sensors in the batch, will be accepted by the *sink* to join its proximity set and the ring will be closed. Alternatively, the *sink* which keeps track of the highest Pid of a node it hears will initiate the closing of the ring. This is not always feasible as the the sensors joining the network are increasingly further apart from the sink; ultimately this effect not affect the either the characteristic path length, or the clustering coefficient of the resulting network. (v) The second period of the self-organization phase starts with event ϵ_{κ_r+1}. Each cluster head computes the integer $v = \lceil 1/p \rceil$ with p the probability in the Watts-Strogatz algorithm and for every $v \times s$ with $s = 1, 2, 3, \ldots$, successor of the sink in the ring attempts to create a shortcut at the time of the events ϵ_{κ_r+s} and requests to join the proximity set of a sensor as close as possible to one "diametrically" positioned in the ring. To do so a cluster head with Pid= k keeps updating the Pid of sensors it could hear during the κ_r events of the first period of the self-organization phase, aiming to get a Pid as close as possible to $k+N/2$ with N the number of sensors in the batch. During this period of self-organization a node grants immediately the request to join its proximity from another sensor diametrically positioned in the ring. Since the Pid are allocated sequentially it is likely that the shortcut

thus constructed will contribute to the reduction of the average path length. Finally, after κ_{sw} events this second period of the self-organization phase finishes. (vi) During the activity phase a sensor examines an *Event List* which contains the indexes of events of interest for the sensor, events when the sensor has to wake up and receive transmissions from the members of its proximity set and then transmit to the sensors whose proximity sets it belongs to.

4.8 Simulation Studies of Self-organizing Sensor Networks

A review of the paper [35] shows that an analytical evaluation of a CTM algorithm is non-trivial and that the analysis of the algorithms described in this dissertation poses significant challenges. We decided to conduct a simulation study and in this section we report on some of our results. A realistic simulation involves a very large number of sensors, of the order of 10^5 and a fairly large number of events, of the order 10^6. Our previous experience shows that network simulators such as $ns-2$ limit the size of the network and the number of events; neither the time-parallel, nor the space-parallel simulation techniques are always feasible and the accuracy of results is questionable [111]. Thus, we decided to implement a simple simulation environment in Java to study the SFSN and the SWAS algorithms.

To generate the distributions of the random variables for the simulation we used the Java library for stochastic simulation (SSJ). Random variables, such as the density, have a normal distribution while others, e.g., the ones controlling the placement of the sensors, have a uniform distribution. We discuss only the results of the simulation for the self-organization phase; the total number of simulation events is equal to κ. Simulation of multiple activity phases assuming a certain sensor failure rate will give us some indication of the life-time of the sensor network and the degradation in time of the ability of sensors to provide accurate information for the entire area covered by the sensor network.

We now discuss the simulation setup and the first issue we address is the placement of the sensors. We consider an area of size $(qa) \times (qa)$, consisting of q^2 elementary squares,

$S_{ij}, 1 \leq i, j \leq q$ of size $a \times a$ with q and a integers, q of the order of 10^1 and a of the order 10^0. The density of sensors has a normal distribution with the mean of ρ and a standard deviation θ. The sensors are placed in individual squares according to the normal distribution mentioned above; the local coordinates x_k, y_k of sensor σ_k placed in the square S_{ij} are random variables uniformly distributed in the range $0 \leq x_k, y_k \leq a$; the global coordinates of this sensor are $X_k = x_k + (i-1)a$ and $Y_k = y_k + (j-1)a$. The total number of sensors in an elementary square is $N_s \approx \rho \times a^2$, so the total number of sensors is $N \approx N_s \times q^2 = \rho \times a^2 \times q^2$.

The average transmission range of a sensor is γ. The average number of sensors able to receive a transmission from a centrally located sensor is $\zeta = \pi \times \gamma^2 \times \rho$; this number is reduced for sensors whose global coordinates satisfy the inequalities: $X_k \ or \ Y_k < \gamma$, or $X_k + \gamma \ or \ Y_k + \gamma > qa$. In our simulation the average density of the sensors ranges from $\rho = 0.9 \times 10^3$ to 1.2×10^3 per elementary square and the standard deviation is $\theta = 50$. We set $a = 1$ and $q = 10$ in our experiments, so the total number of sensors varies from about $N \approx 0.9 \times 10^5$ when $\rho = 0.9 \times 10^3$ to $N \approx 1.2 \times 10^5$ when $\rho = 1.2 \times 10^3$. In our simulation the probability of splitting for the MAC algorithm is $p = 0.5$. The probability that the fitness function of a sensor returns "true" is 0.9.

4.8.1 Simulation Study of the SFSN Algorithm

The algorithm for self-organization does not guarantee that every sensor will be a node in the random graph constructed during the self-organization phase, nor does it guarantee that each node included in this random graph will be connected with precisely μ other nodes. Sensors whose actual cardinality of the proximity set is equal to zero are isolated; the sensors σ_i such that $\mid \mathcal{P}(\sigma_i) \mid << \mu$ may not be able to have their information disseminated in the presence of sensor failures. Therefore, an objective of the simulation study is to construct a histogram of the cardinality of proximity sets of all sensors at the end of the self-organization phase.

A fair number of parameters affect the system; among them we note the transmission range, γ, the average density of the sensors, ρ, and the maximum cardinality of the proximity set μ.

Figures 4.7 (a), (c), and (e) show the effect of γ, the transmission range, on μ_{actual}, the actual number of sensors in the proximity set, when $\mu = 10$. As the number of events increases from 0.5×10^6 to 1.0×10^6 and then to 1.5×10^6 the percentage of sensors that are isolated ($\mu_{actual} = 0$) decreases from $(3 - 5)\%$ to zero. At the same time, the percentage of sensors able to fill out their proximity set to the maximum value, $\mu = 10$, increases from $(8 - 12)\%$ to $(55 - 60)\%$. The histograms show that when γ is in the range $0.09a - 0.12a$ then only a small fraction of 1% percent of the sensors are isolated and very few of them have a proximity set with 2 or fewer sensors, regardless of the transmission range. A small variation of the transmission range does not affect the random graph constructed as a result of the self-organization phase in any significant way provided that the number of events κ is at least one order of magnitude larger than the number of sensors. This shows that the algorithm is robust, it is able to accommodate the reduction of the transmission range due to depletion of power reserves of individual sensors. Figures 4.7 (b), (d), and (e) show the effect of the maximum cardinality of the proximity set on the actual number of sensors in the proximity set. As the number of events increases from 0.5×10^6 to 1.0×10^6 and then to 1.5×10^6 a larger percentage of sensors are able to fill up their proximity sets. The histograms show that for $\mu = 4$, the fraction of sensors able to fill-up their proximity set increases from about 60% to 85%; when $\mu = 6$ the fraction increases from about 38% to about 80% and when $\mu = 10$ the fraction increases from about 10% to about 52%.

From the results in Figures 4.7 we conclude that the number of events in the simulation should be at least one order of magnitude larger than the number of sensors. Of course, even in our relatively simple and efficient simulation environment a simulation covering more that 1.5×10^6 events is computationally expensive. We have to balance the accuracy of

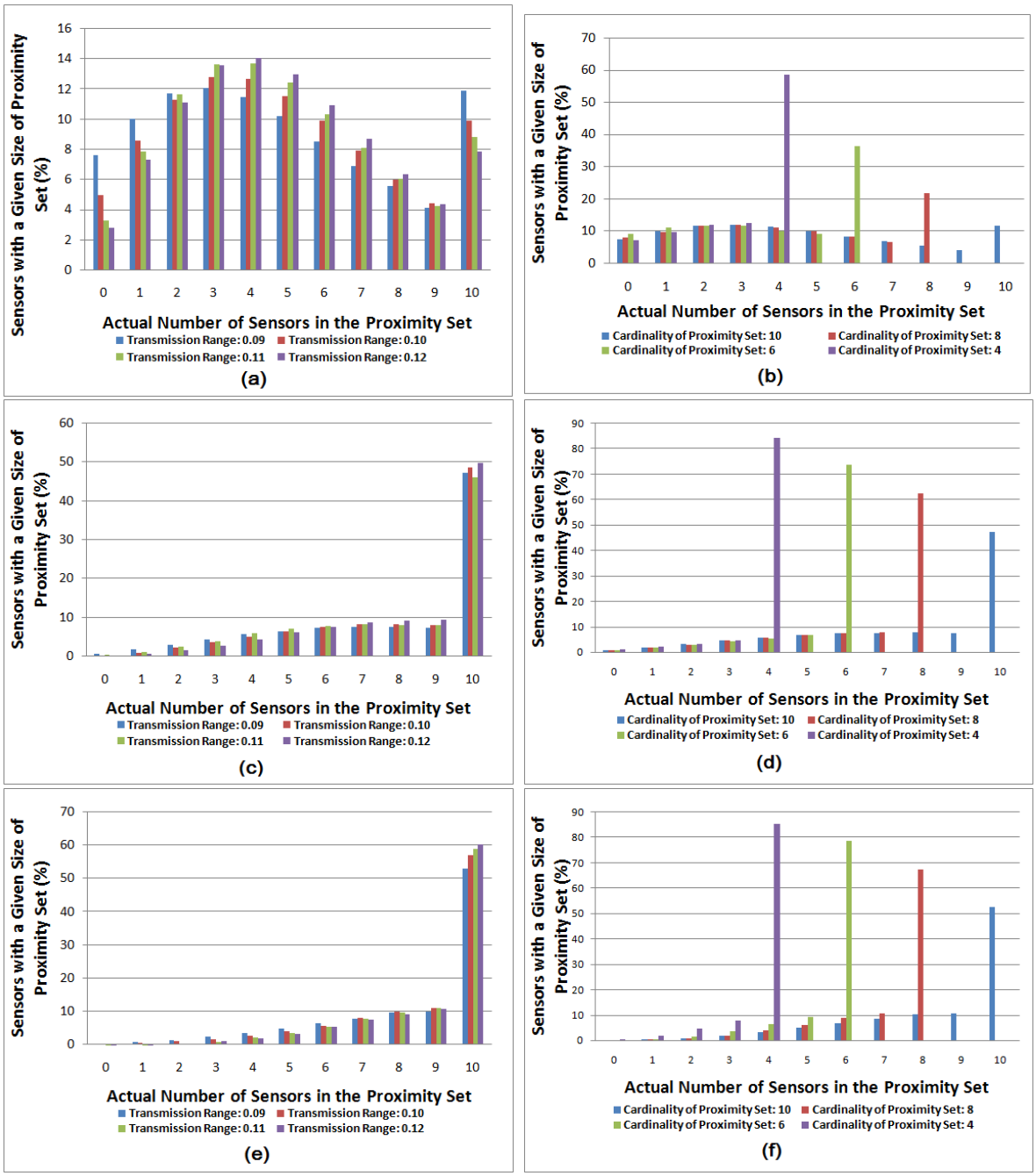


Figure 4.7: The effect of the transmission range (γ) and of the maximum cardinality of the proximity set (μ) on the actual number of sensors in the proximity set (μ_{actual}). $N \approx 10^5$ and $\rho = 1 \times 10^3$. The number of events per phase is: $\kappa = 0.5 \times 10^6$ in (a) and (b), $\kappa = 1.0 \times 10^6$ in (c) and (d), and $\kappa = 1.5 \times 10^6$ in (e) and (f). The histograms in (a), (c), and (e) show the effect of γ with $\gamma = 0.09a, 0.1a, 0.11a, 0.12a$. The histograms in (b), (d), and (e) show the effect of μ when the average transmission range is $\gamma = 0.09a$ and $\mu = 4, 6, 8, 10$.

the simulation results and the simulation cost and have decided to limit our experiments to 1.5×10^6 events.

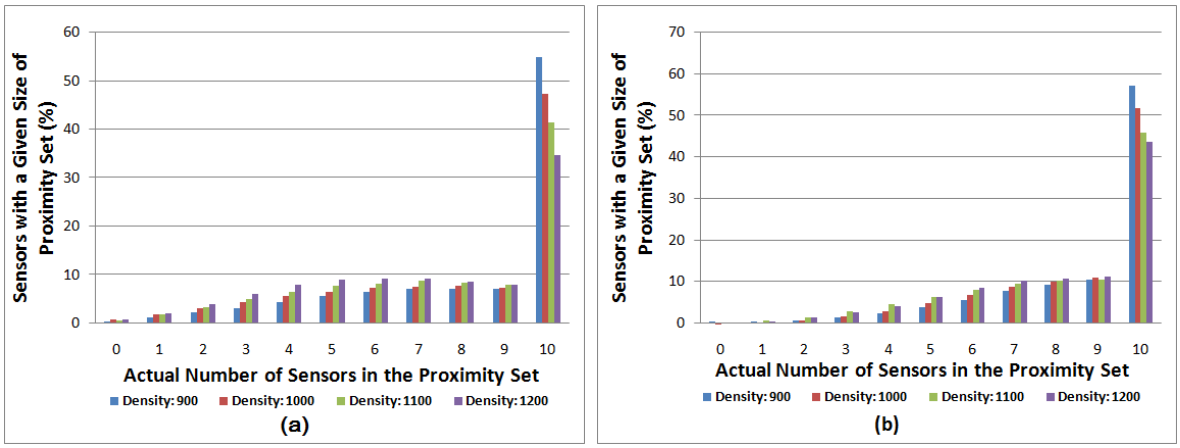


Figure 4.8: Histograms showing the effect of the average sensor density (ρ) upon the actual number of sensors in the proximity set (μ_{actual}) for (a) $\kappa = 1.0 \times 10^6$ and (b) $\kappa = 1.5 \times 10^6$. The maximum cardinality of the proximity set is set to $\mu = 10$ and the transmission range is set to $\gamma = 0.09a$. The average density is $\rho = (0.9, 1.0, 1.1, 1.2) \times 10^3$. The total number of sensors is in the range $N \approx 0.9 \times 10^5 - 1.2 \times 10^5$.

The effect of the average sensor density on the actual number of sensors in the proximity set is presented in Figures 4.8 (a) when $\kappa = 1.0 \times 10^6$ and (b) when $\kappa = 1.5 \times 10^6$. We observe that the density affects μ_{actual}. Indeed, when $\kappa = 1.0 \times 10^6$ then the percentage of sensors with $\mu_{actual} = 10$ ranges from a low of about 35% for $\rho = 1.2 \times 10^3$ to a high of about 55% for $\rho = 0.9 \times 10^3$. When $\kappa = 1.5 \times 10^6$ the range is slightly smaller: 42% to 55%. Increasing the average density from 0.9×10^3 to 1.2×10^3 causes the average number of sensors that could potentially respond to an RtJ to increase by almost 40%, from an average of 26 to 35. Of course, as the algorithm progresses many sensors do not respond to an RtJ due to the limitations imposed by μ and M.

Another interesting question is the distribution of the reciprocal events recorded by individual sensors; these events control the time during the activity phase when each sensor wakes up to receive communication from the sensors which have included the sensor in their

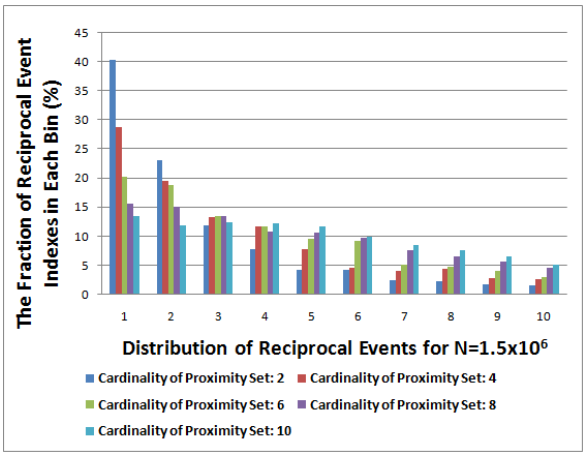


Figure 4.9: Histogram showing the distribution of the reciprocal event indices which control the time during the activity phase when each sensor wakes up to receive communication and then transmits.

proximity sets and then transmits to the sensors in their own proximity set. In our simulation we consider 10^5 sensors and 1.5×10^6 events and experiment with several values for the maximum size of the proximity set, $\mu = 2, 4, 6, 8, 10$. The events are grouped into 10 bins and we record the percentage of the reciprocal events in each bin. The graph in Figure 4.9 shows that a larger size of the proximity set provides a more uniform distribution of the reciprocal events during the activity phase. For example, when $\mu = 10$ then the distribution of the indices of the reciprocal events is: 13% in the range $1 - 150,000$; 12% in the range $150,001 - 300,000$; 12% in the range $300,001 - 450,000$; 12% in the range $450,001 - 600,000$; 11.5% in the range $600,001 - 750,000$; 10.0% in the range $750,001 - 900,000$; 8% in the range $900,001 - 1,050,000$; 7.5% in the range $1,050,001 - 1,200,000$; 6.5% in the range $1,200,001 - 1,350,000$; and 5% in the range $1,350,001 - 1,500,000$.

4.8.2 Simulation Study of the SWAS Algorithm

Call $n \leq N$ the *Pid* of the last node connected to sensor network, be it cluster head or a member of cluster. If that the actual placement of the sensors allows each cluster head to have precisely μ sensors in its cluster, then the size of a cluster including the cluster head

(the node in the ring) is $\mu + 1$. Then n_r, total number of nodes in the ring proper and n_0, the total number of nodes connected to the nodes in the ring proper are respectively:

$$n_r = \frac{n}{\mu + 1} \quad and \quad n_o = n - n_r = n\frac{\mu}{\mu + 1}.$$

The coverage of the algorithm is defined as the ratio of sensors connected to the network to the total number of sensors $c = n/N$. The characteristic path length L_p is defined as the number of edges in the shortest path between two sensors in the directional logical ring constructed, averaged over all pairs of sensors connected to the network. The clustering coefficient C_p is computed as the average node degree over all sensors N.

We now discuss the simulation setup and the first issue we address is the placement of the sensors. We consider an area of size $(qa) \times (qa)$, consisting of q^2 elementary squares, $S_{ij}, 1 \leq i, j \leq q$ of size $a \times a$ with q and a integers, q of the order of 10^1 and a of the order 10^0. The density of sensors has a normal distribution with the mean of $\rho = 0.5$ and a standard deviation θ. The sensors are placed in individual squares according to this normal distribution; the local coordinates x_k, y_k of sensor σ_k placed in the square S_{ij} are random variables uniformly distributed in the range $0 \leq x_k, y_k \leq a$; the global coordinates of this sensor are $X_k = x_k + (i-1)a$ and $Y_k = y_k + (j-1)a$. The total number of sensors in an elementary square is $N_s \approx \rho \times a^2$. The average transmission range of a sensor is γ. The average number of sensors able to receive a transmission from a centrally located sensor is $\zeta = \pi \times \gamma^2 \times \rho$; this number is reduced for sensors whose global coordinates satisfy the inequalities: $X_k \ or \ Y_k < \gamma$, or $X_k + \gamma \ or \ Y_k + \gamma > qa$. In our simulation we set $a = 1$ and $q = 10$; the average density of the sensors is $\rho = 10^4$ per elementary square and the total number of sensors is $N = 10^6$. The probability of splitting for the splitting algorithm is $p_s = 0.5$. The probability that the fitness function of a sensor returns the value "true" is 0.9.

The simulation discussed in this dissertation covers only the self-organization phase; the total number of events covered is $\kappa = 2 \times 10^6$. Indeed, the formation of the ring requires

a number of events equal to the total number of sensors and then only a fraction of cluster heads on the ring attempt to create shortcuts. During the first period of the self-organization phase the algorithm leads to the creation of a ring connecting the cluster heads. Each cluster head σ_i has attached to it μ sensors in the cluster $\mathcal{C}_i$; during the activity phase the cluster head collects information from the sensors in its cluster.

A first question answered by our simulation study regards the coverage of the algorithm. Figure 4.10 shows the percentage of sensors covered function of the transmission range and function of the cluster size. The transmission rage is measured relative to the size of the elementary squares used for the random placement of sensors; for example, a range of 0.2 means that the transmission range of a sensor is $\gamma = 0.2a$ where a^2 is the area of an elementary rectangle. When the cluster size is $\mu + 1 = 20$ and the transmission range is $\gamma = 0.4a$, 98% of the sensors are connected to the network; when the transmission range is reduced to half of its original value, namely to $\gamma = 0.2a$, then this ratio decreases to 85% thus, the ability of the algorithm to cover a large percentage of the sensors is resilient to changes in the transmission range. The transmission range could decrease in time as the power reserves of the sensors are depleted; the results summarized in Figure 4.10 (Top) give us confidence that the sensor network constructed using the SWAS algorithm satisfies one of the desiderates of a VLSN, to be operational over long periods of time.

We are also interested in the effect of the cluster size upon coverage. We assume a transmission range $\gamma = 0.3a$ and allow cluster size to increase five fold from 20 to 100; under these conditions the coverage decreases from 94% to 75%, Figure 4.10 (Bottom). This somewhat surprising effect is due to the fact that there is a limited number of sensors in the vicinity of a cluster head σ_i; once they are included in the cluster $\mathcal{C}_i$, there are fewer candidates to become cluster heads and join the ring. Increasing the sensor density above the current value of $\rho = 10^4$ per elementary square will most likely counter this undesirable effect. Yet, increasing the sensor density will lead to an increase of the number of collisions during the first period of the self-organization phase, the formation of ring.

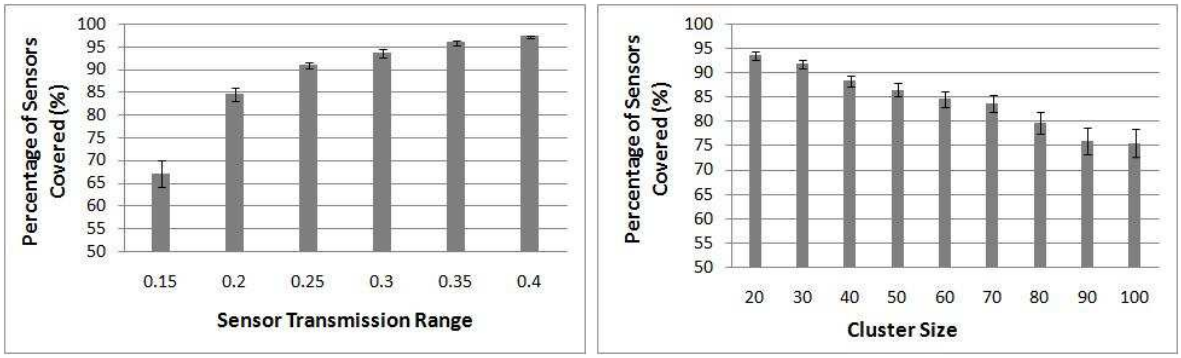


Figure 4.10: Fraction of sensors connected to VLSN function of: (Top) the transmission range, γ when the size of individual clusters is $\mu = 20$; (Bottom) the cluster size when the transmission range is $\gamma = 0.3a$. 95% confidence interval is shown.

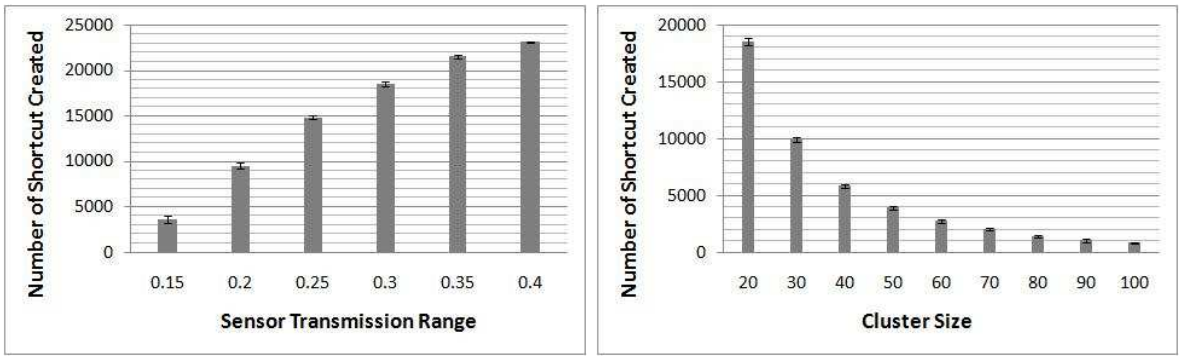


Figure 4.11: The number of shortcuts function of: (Top) the transmission range γ when the size of individual clusters is $\mu = 20$; (Bottom) the cluster size when the transmission range is $\gamma = 0.3a$. 95% confidence interval is shown.

To create a shortcut the sensor σ_i with $Pid = k \leq N/2$ attempts to connect to a node diametrically located in the ring. The sensor searches for a partner among the sensors with a Pid in the range $k+0.9\times N/2$ to $k+1.1\times N/2$. To do so the sensor with $Pid = k$ monitors the transmissions of nodes in the target range and records the Pid of the the first sensor in this range whose transmissions it can hear; then improves on its choice if the physical placement of subsequent sensors in this range permits. First, we assume that every cluster head attempts to create a shortcut, $s = \lceil p \rceil = 1$. Some fail to reach a node "diametrically" positioned in the logical ring and, as we shall see shortly, less than half of the cluster heads are successful.

We study the evolution of the number of shortcuts function of the transmission range for a given cluster size, $\mu = 20$ and function of the cluster size when the transmission range is fixed, $\gamma = 0.3a$; Figure 4.11 summarizes our findings. As expected, when the transmission range increases in the range $0.15a \leq \gamma \leq 0.4a$ the number of shortcuts increases from about $3,500$ to more than $23,000$, Figure 4.11 (Top). The cluster size has a significant effect upon the number of shortcuts as the number of cluster heads decreases, Figure 4.11 (Bottom). When $\mu = 20$ there are roughly $850,000/20 = 42,500$ clusters and we have some $18,000$ shortcuts. For $\mu = 100$ there are roughly $750,000/100 = 3,750$ clusters and there are about $1,000$ shortcuts.

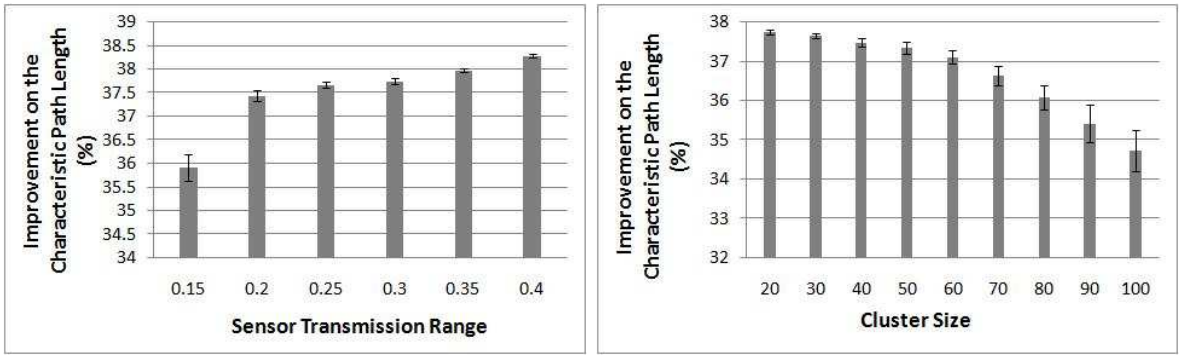


Figure 4.12: Relative improvement of the characteristic path length due to the shortcuts function of: (Top) the transmission range γ when the size of individual clusters is $\mu = 20$; (Bottom) the cluster size when the transmission range is $\gamma = 0.3a$. 95% confidence interval is shown.

An important argument in favor of the small-worlds topologies is the reduction of the characteristic path length. Figure 4.12 shows the improvement of the characteristic path length L function of the transmission range and function of the cluster size. We see a reduction of about 37.5% of the characteristic path length over the interval $0.15a \leq \gamma \leq 0.4a$ of transmission range values, Figure 4.12 (Top). This result is encouraging and shows that the algorithm is robust in terms of its ability to reduce the characteristic path length when the power reserves are depleted and the transmission range shrinks.

The effect of the cluster size upon the characteristic path length when the transmission rage is $\gamma = 0.3a$ is presented in Figure 4.12 (Bottom). When the cluster size increases five-fold from $\mu = 20$ to $\mu = 100$ we notice a slight reduction of the improvement from about 37.5 when $\mu = 20$ to about 34.6% for $\mu = 100$. We conclude that the algorithm allows us to extend the cluster size without a substantial penalty in the average characteristic path length.

The clustering coefficient $C(p)$ is only slightly affected by the creation of shortcuts. The clustering coefficient increases by about 0.75% when the transmission range is $\gamma = 0.25a$; it increase by 2.5% when $\gamma = 0.5a$. When $\gamma = 0.3a$ and the cluster size is in the range $20 \leq \mu \leq 100$ the clustering coefficient increases due to addition of shortcuts by about 1.5%.

Another goal of our preliminary investigation of the algorithm is to study L_p and C_p for $0 \leq p \leq 1$. As noted earlier, a cluster head may or may not be able to identify a target to a shortcut subject to conditions which guarantee a range of distance between the two *Pid*s. Rather than allowing each cluster head to attempt to create a shortcut with probability p, we define a *shortcut index* s; when $s = 1$, then every cluster head attempts to create a shortcut; when $s = 2$ every other cluster head attempts to create a shortcut, and so on. Figure 4.13 shows the effect of this selection process when $s = 4, 8, 16, 32, 64, 256, 512, 1024, 2048$. Surprisingly, the improvement of characteristic path length does not fall below 35% when $s < 512$, Figure 4.13 (Top). The effect upon the clustering coefficient is negligible, Figure 4.13 (Bottom).

4.9 Conclusions

Self-organization is a trait of adaptive systems with limited resources, it ensures scalability and confers the ability to survive. The SFSN algorithm supports free-scale, self-organizing sensor networks. It supports reorganization at discrete moments of time controlled by a random sequence of events. This primitive form of self-organization is more suitable for

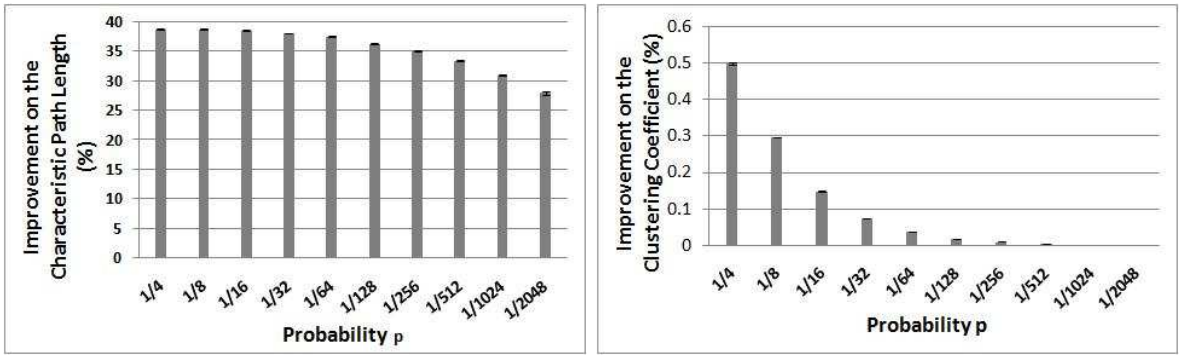


Figure 4.13: (Top) Relative improvement of the characteristic path length; (Bottom) Relative change of the clustering coefficient due to the shortcuts function of the shortcut index s. The size of individual clusters is $\mu = 20$ and the transmission range is $\gamma = 0.3a$. 95% confidence interval is shown.

low-cost devices with limited power and storage capacity than a continuous self-organization scheme that requires a fair amount of state and past history information [76].

During a self-organization phase the SFSN algorithm constructs a random graph connecting a sensor with a bounded number of other sensors. Each sensor σ builds a proximity set, $\mathcal{P}_\sigma$, with at most μ members. For each member of the proximity set sensor σ stores at most M event indices; these events are used during the following activity phases to determine when sensor σ should wake up to transmit and receive information from the members of $\mathcal{P}_\sigma$. Even though all sensors use the same "wake-up to receive and then transmit" pattern during all activity phases, the actual times and frequencies used for communication are driven by the random number generators, thus are unpredictable for an external observer.

The construction of the proximity sets of bounded cardinality, $\mathcal{P}_\sigma \leq \mu$, is a challenging task because the sensors are anonymous. We use the collision resolution algorithm to attribute a unique Pseudo-Id (Pid) to each sensor and thus limit the number of sensors in a proximity set; this Pid plays no role during the activity phase when sensors are only aware of the indices of the slots when they are expected to wake up and either send or receive a message. The communication schedule followed by all sensors prevents collisions during the activity phases and minimizes the time when the sensors use their RF sub-system to

send or receive messages; thus, it minimizes the power consumption of individual sensors and extends the lifetime of the sensor network. The system provides information assurance with minimal power consumption and avoids expensive encryption/decryption techniques discussed in the literature; the time and the frequency used for communication are random numbers and no external entity can either join in, or predict the time when the sensors in the set will transmit and attempt to interfere with the transmissions.

The algorithm for self-organization does not guarantee that all sensors will be included in the random graph, or that each sensor will be actually connected with μ other sensors. Yet our simulation results are encouraging; a small fraction of sensors are isolated, while a very large fraction of them are able to fill out their proximity sets. The fact that the density does not have a significant effect upon the system is encouraging, it leads us to believe that the system is rather resilient to failure. Indeed, reducing the sensor density approximates sensor failures to some extent. Also the histogram of the actual number of sensors in the proximity set shows that a small variation of the transmission range does not affect the random graph constructed as a result of the self-organization phase in any significant way; this shows that the algorithm is robust, it is able to accommodate the reduction of the transmission range due to depletion of power reserves of individual sensors.

Our simulation results also show that a larger size of the proximity set provides a more uniform distribution of the reciprocal events during the activity phase. A simulation similar to the one reported in this dissertation is always necessary to determine the number of events in an epoch prior to the actual deployment. Based upon these results we conclude that our algorithm has some advantages over more traditional schemes for organization of sensor networks.

With SFSN algorithm, we further develop the SWAS algorithm. The SWAS algorithm allows us to create sensor networks with a topology approximating small-worlds networks. The networks are not in danger to be disconnected when a sensor fails, have a relatively short average path length, communicate optimally (in other words establish collision-free

communication channels); the algorithm constructs a schedule that minimizes the time when each sensor has to wake up and transmit or receive while the sensors maintain a very limited amount of information.

During the self-organization phase the sensors form clusters and the cluster heads are interconnected into a regular topology, a directed graph rooted at the sink; then, to construct a small worlds network a cluster head creates a shortcut to a diametrically located cluster head. Simulation results show that a very large fraction of all sensors (in some cases 97%) are able to join the VLSN and this fraction is affected by the cluster size and by the transmission range The number of shortcuts increases with the transmission range and decreases substantially when the cluster size increases. The algorithm supports a substantial reduction of the characteristic path length.

CHAPTER 5: SUMMARY AND FUTURE WORK

Distributed resource management is the key to deploying the computing efforts in a large-scale distributed environment. Effective resource management strategies and algorithms ensure Quality of Service (QoS) and scalability. In this dissertation, we focus on multiple levels of effective resource management techniques.

We first consider a classical resource management problem, namely the scheduling of data-intensive applications. The throughput of computation and the execution time of the data-intensive applications can be greatly improved by allocating the workload to multiple computing resources for parallel processing. A large-scale distributed computing platform, such as the Grid, characterized by the heterogeneity of the computation resources, high variability of resource availability, expensive data transfer cost, and high coordination overhead poses great challenges on scheduling a divisible load task consisting of multiple interdependent task instances.

We define the *Divisible Load Scheduling (DLS)* problem, introduce the optimal divisible load scheduling and the related fault-tolerant coordination algorithms suitable for the data-intensive applications with divisible load. The divisible load scheduling model discussed in this dissertation is based on the assumption that data staging and all communication with the sites can be done in parallel, as opposed to the one-port model where data staging is strictly sequential. We define four performance evaluators: the available time, the execution rate, the duty cycle, and the data transfer rate, to characterize the heterogeneity of the computing resources. Different data staging strategies are discussed and the DLS algorithms for each of them are presented with the proof of the optimality of the solution. The DLS algorithms we introduced exploit parallel communication, consider realistic scenarios regarding the time when target systems are available, and generate optimal schedules. Performance studies show that these algorithms perform better than divisible load scheduling algorithms based upon sequential communication.

We note that while the DLS algorithms discussed in this dissertation are optimal, they require information that is rarely accurate. Indeed, neither the available time and duty cycle, nor the execution rate of a program on a specific target system, or the data transfer rate, can be determined with high accuracy. An effective strategy to improve the accuracy of the estimation of the resource parameters is to exploit the feature of the PDS model. After each pipeline stage, the processing rate, duty cycle and data transfer rate on each restricted target system can be updated based on practical measurement and the FLX-PDS algorithm is run again on the restricted target systems set to generate a new data partition. This parameter update process can happen after each pipeline stage or after every several pipeline stages which provides the "self-adaptation" capability for the DLS. The future work also involves designing a system to automatically monitor the resource status and record post-mortem data about individual executions. An analysis of these history records will allow us to better estimate the performance evaluators for each target system.

On a higher level, the method of organizing and managing the large amount of computing resources over the computing network to reduce the resource consumption and improve the quality of service is another concern of this dissertation.

We have developed a model of self-organization for a complex computation and communication system inspired by biological metaphors that uses the concept of varying energy levels to express activity and goal satisfaction. It is assumed that intentionality expressed at the application level would be mapped to these energy values and manipulated by the actions of genes. These genes, in turn, capture the system characteristics and express behavior. The system was simulated and experiments that were explicitly compared to specific application domains were performed. The self-organization property was verified. Although promising, these results are preliminary and further studies are needed to draw general conclusions.

Here, we did not consider reorganization, which is a critical aspect of the life-cycle of entities. In that scenario, entities that do not perform well as measured by our success

metric should stop their activities, recharge their active material and then attempt to form and/or to join new or existing organizations.

The model introduced seems to provide enough flexibility to allow us to consider conflicting common, class, and individual goals but requires fine tuning and more testing before we can assess its applicability to other application areas. A practical example of a conflicting goal is when the common goal aims to reduce the carbon footprint, the class goal is to achieve a certain level of QoS (e.g., to meet the deadlines, the class goal of producers is to maximize their benefits), and the entity specific goal is to improve its effective kinetic to kinetic energy ratio.

The future work includes extending the simulation results and considering different sets of parameters. To encoding application intentionality as part of the model is a vast area that also requires attention.

The self-organization model for complex computing and communication systems is further applied to Very Large Sensor Networks (VLSNs). An algorithm for self-organization of anonymous sensor nodes called SFSN (Scale-free Sensor Networks) is introduced.

The SFSN algorithm is designed for VLSNs consisting of a fairly large number of inexpensive sensors with limited resources yet, expected to function over prolonged periods of time. An important feature of the algorithm is the ability to interconnect sensors without an identity, or physical address used by traditional communication and coordination protocols. The role of the self-organization phase is to create collision-free communication channels allowing a sensor to synchronously forward information to the members of its proximity set during the activity phases that follow. The self-organization phase is started by a sink node which requests to join the proximity set of one of its neighbors.

The SFSN ensures the scalability, the number of sensors each sensor communicates with and the amount of state information each node has to maintain are strictly limited regardless of the total number of sensors in the network; a node in the random graph showing the system's connectivity is linked with at most a pre-determined number of nodes, thus the

system is scale-free. This scheme limits the amount of communication and the complexity of coordination.

With SFSN, we further introduce the SWAS (Small-Worlds of Anonymous Sensors) algorithm. The SWAS algorithm introduced in this dissertation is unique in its ability to create a sensor network with a topology approximating small-worlds networks. Rather than creating shortcuts between pairs of diametrically positioned nodes in a logical ring we end up with something resembling a double-stranded DNA. Experiments show that the characteristic path length and the clustering coefficient are not affected when the actual placement of the sensors does not allow the closing of the ring which supports unidirectional communication during the activity phase.

A major problem of the algorithm is the serial nature of the process to construct the ring of clusters. We now investigate a version of the algorithm to speed-up the process; in this version the establishment of a cluster is done as a result of a single collision resolution interval (CRI) that starts with the cluster head requesting to be accepted as a member of the proximity set of another sensor. A new strategy to attribute Pids to sensors must be considered; cluster heads could then get consecutive Pids and cluster members will get a local Pid based upon the order they succeed to transmit during the CRI. Then, during the activity phase, all members of a cluster could report either sequentially, when the cluster head is allocated one slot to collect information, or in parallel, using multiple frequency channels if the hardware permits.

REFERENCES

[1] R. Abbott. *Complex Systems Engineering: Putting Complex Systems to Work.* Complexity, Vol.13, No. 2, pp.10–11, 2007.

[2] I. Akyildiz, W. Su, Y. Snakarasubranmaniam, and E. Cayirci. *A Survey on Sensor Networks.* Computer Networks, Vol.40, pp.102–114, 2002.

[3] D. Altilar and Y. Paker. *An Optimal Scheduling Algorithm for Parallel Video Processing.* In IEEE Int. Conference on Multimedia Computing and Systems (ICMCS'98), pp.245, 1998.

[4] S. Aluru, N. Amato, D.A. Bader, S. Bhandarkar, L. Kale, and D.C. Marinescu. *Parallel Computational Biology.* Parallel Processing for Scientific Computing, Software, Environments, and Tools, Vol.20, pp.356–378, SIAM, 2006..

[5] M.J. Atallah, C.L. Black, D.C. Marinescu, H.J. Siegel, and T.L. Casavant. *Models and Algorithms for Co-scheduling Compute-intensive Tasks on a Network of Workstations.* Journal of Parallel and Distributed Computing, Vol. 16, No. 4, pp.319–327, 1992.

[6] X. Bai, D.C. Marinescu, L.Bölöni, H. J. Siegel, R.E. Daley, and I-J. Wang. *A Macroeconomic Model for Resource Allocation in Large-scale Distributed Systems.* Journal of Parallel and Distributed Computing, Vol.68, pp.182–199, 2008.

[7] D.J. Baker and A. Ephemides. *The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm.* IEEE Trans. Comm, Vol.11, pp.1694–1701, 1981.

[8] A-L. Barabási and R. Albert. *Emergence of Scaling in Random Networks.* Science, Vol.286, No.5439, pp.509–512, 1999.

[9] A-L. Barabási, R. Albert, and H. Jeong. *Scale-free Theory of Random Networks; the Topology of World Wide Web.* Physica A, Vol.281, pp.69–77, 2000.

[10] R. Baraglia, R. Ferrini, N. Tonellotto, L. Ricci, and R. Yahyapour. *A Launch-time Scheduling Heuristics for Parallel Applications on Wide Area Grids.* Journal Of Grid Computing, Vol.6, pp.159–175, 2008.

[11] S. Bataineh and T.G. Robertazzi. *Distributed Computation for a Bus Network with Communication Delays.* Proc. Conf. Information Sciences and Systems, Baltimore, MD, pp.709–714, 1991.

[12] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. *Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems.* IEEE Trans. Parallel Distributed Systems, Vol.16, pp.207–218, 2005.

[13] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Computer Society Press, ISBN:0818675217, 1996.

[14] V. Bharadwaj, D. Ghose, and T.G. Robertazzi. *Divisible Load Theory: A new Paradigm for Load Scheduling in Distributed Systems.* Cluster Computing on Divisible Load Scheduling, vol.6, No.1, pp.7–18, 2003.

[15] J. Blazewicz, M. Drozdowski, and M. Markiewicz. *Divisible Task Scheduling - Concept and Verification.* Parallel Computing, Vol.25, pp.87–98, 1999.

[16] J. Blazewicz and M. Drozdowski. *Scheduling Divisible Jobs on Hypercubes.* Parallel Computing, Vol.21, pp.1945–1956, 1995.

[17] J. Blazewicz and M. Drozdowski. *The Performance Limits of a Two-Dimensional Network of Load-Sharing Processors.* Foundations of Computing and Decision Sciences, Vol.21, No.1, pp.3–15, 1996.

[18] L.Bölöni and D.C. Marinescu. *Robust Scheduling of Metaprograms.* Journal of Scheduling, Wiley, Vol.5, No.5, pp.395–412, 2002.

[19] V. Bharghavan, A. Demers, S. Shenker, L. Zhang. *MACAW: a Medium Access Protocol for Wireless LAN's.* Proc. ACM SIGCOMM Conference (SIGCOMM'94), pp.212–225, 1994.

[20] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Inspiration for Optimization from Social Insect Behavior.* Nature, Vol.406, pp.39–42, 2000.

[21] D. Braginsky and D. Estrin. *Rumor Routing Algorithm for Sensor Networks.* Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications, pp.22–31, 2002.

[22] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund. *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems.* Journal of Parallel and Distributed Computing, Vol.61, No.6, pp.810–837, 2001.

[23] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments.* In Proceedings of the 9th Heterogeneous Computing Workshop (HCW00), pp.349–363, 2000.

[24] S.F. Camazine, J.-L. Deneubourg, N.R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-organization in Biological Systems.* Priceton University Press, Princeton, NJ, ISBN: 06910121132001, 2001.

[25] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. *SPAN: an Energy Efficient Coordination Algorithm for Topology Maintenance in Adhoc Wireless Networks.* ACM Wireless Networks Journal, Vol.8, No.5, pp.481–494, 2002.

[26] Y.-C. Cheng and T.G. Robertazzi. *Distributed Computation with Communication Delay.* IEEE Transactions on Aerospace and Electronic Systems 24, pp.700–712, 1988.

[27] Y.-C. Cheng and T.G. Robertazzi, *Distributed Computation for a Tree Network with Communication Delays*. IEEE Trans. Aerospace and Electronic Systems, vol.26, No.3, pp.511–516, 1990.

[28] J.O. Kephart and D.M. Chess *The Vision of Autonomic Computing.* IEEE Computer, Vol.36, pp.41–50 2003.

[29] T. C. Collier and C. Taylor. *Self-organization in Sensor Networks.* Journal of Parallel and Distributed Computing, Vol.64, No.7, pp.866–873, 2004.

[30] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley, New York, NY, ISBN: 0471241954, 1991.

[31] J.P. Crutchfield and J.P. Shalizi. *Thermodynamic Depth of Causal States: Objective Complexity via Minimal Representation.* Physical Review, Vol.59, pp.275–283, 1999.

[32] F. Darema-Rodgers, V.A. Norton, and G.F. Pfister. *Using A Single-Program-Multiple-Data Computational Model for Parallel Execution of Scientific Applications*. Technical Report RC11552, IBM T.J Watson Research Center, 1985.

[33] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry. *Epidemic Algorithms for Replicated Database Maintenance.* Proc. 6th ACM Symp. on principles of distributed computing, pp.1-12, 1987.

[34] F.C. In, K.S. Kim, H. Okagawa, K. Shrikh, and L.G. Kazovsky. *Unslotted Optical CSMA/CA MAC Protocol with.* Wireless Personal Communications, Vol.11, pp.11–161, 1999.

[35] G. Fayolle, P. Flajolet, M. Hofri, and P. Jacquet. *Analyis of a Stack Algorithm for Random Multiple-access Communication.* IEEE Transaction on Information Theory, Vol.31, No.2, pp.244–254, 1985.

[36] R.P. Feynman, R.B. Leighton, and M. Sands *The Feynman Lectures on Physics.* Addison-Wesley, Reading, MA, ISBN: 0201021153, 1977.

[37] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, ISBN: 1558604758, 2000.

[38] R. Gallagher. *A Perspective on Multiaccess Channels.* IEEE Transaction on Information Theory, Vol.31, No.2, pp.124–142, 1985.

[39] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. *Highly Resilient, Energy Efficient Multipath Routing in Wireless Sensor Networks.* Mobile Computing and Communications Review (MC2R), Vol.5, No.4, pp.11–25, 2002.

[40] M. Gell-Mann. *Simplicity and Complexity in the Description of Nature.* Engineering and Sciences, California Institute of Technology, Vol.3, pp.3–9, 1988.

[41] M. Gerla and J.T. Tsai. *Multicluster, Mobile, Multimedia Radio Networks.* Wireless Networks, pp.255–265, 1995.

[42] C. Gershenson and F. Heylighen. *When Can We Call a System Self-organizing.* Advances in Artificial Life, Vol.2801, pp.606–614, Springer Verlag, 2003.

[43] A. Giersch, Y. Robert, and F. Vivien. *Scheduling Tasks Sharing Files from Distributed Repositories.* Technical Report RR-2004-04, LIP, ENS Lyon, France, 2004.

[44] J.D. Halley and D.A. Winkler. *Classification of Emegence and its Relation to Self-organization.* Complexity, Vol.13, No.5, pp.10–15, 2008.

[45] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu . *The Autonomic Computing Paradigm.* Cluster Computing: The Journal of Networks, Software Tools, and Applications, Vol.9, No.1, pp.5–17, 2006.

[46] F. Heylighen, and C. Gershenson. *The Meaning of Self-organization in Computing.* IEEE Intelligent Systems, pp.72–75, 2003.

[47] J. Hopfield. *Neural Networks and Physical Systems with Emergent Collective Computational Abilities.* Proc. National Academy of Science, Vol.79, pp.2554–2558, 1982.

[48] Q. Hong and J. Ju. *Cooperative Task Scheduling on Workstations Network.* Journal of Software, Vol.9, No.1, pp.14–17, 1998.

[49] Q. Huang, J. Cukier, H. Kobayashi, and J. Zhang. *Fast Authentication Key Establishment Protocols for Self-organizing Sensor Networks.* Proc. Wireless sensor networks and Applications (WSNA'03), pp.141–150, 2003.

[50] A.W. Hübler. *Understanding Complex Systems.* Complexity, Vol.12, No.5, pp.9–11, 2007.

[51] *IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* IEEE-SA, 2007.

[52] *iperf.* http://dast.nlanr.net/projects/Iperf/

[53] V. Jacobson. *Congestion Avoidance and Control.* Proceedings of ACM SIGCOMM '88, pp.314–329, ISBN:0897912799, 1988.

[54] Y. Ji, D.C. Marinescu, W. Zhang, X. Zhang, X. Yan, and T.S.Baker. *A Model-based Parallel Origin and Orientation Refinement Algorithm for CryoTEM and its Application to the Study of Virus Structures.* Journal of Structural Biology, Vol.154, No.1, pp:1–19, 2006.

[55] M. Joa-Ng. *Spread Spectrum Medium Access Protocol with Collision Avoidance using Controlled Time of Arrival.* Telecomunication Systems, Vol.18, pp.169–191, 2001.

[56] H.D. Karatza. *Gang Scheduling and I/O Scheduling in a Multiprocessor System.* Proc. Symp. on Performance Evaluation of Computer and Telecommunication Systems (SCSI), pp.245–252, 2000.

[57] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks.* Wiley, ISBN: 0470095105, 2005.

[58] C. Karlof, Y. Li, and J. Polastre. *ARRIVE: Algorithm for Robust Routing in Volatile Environments.* Technical Report UCB/CSD-03-1233, U.C. Berkeley, 2002.

[59] G. Khanna, N. Vydyanathan, T. Kurc, U.V. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. *A hypergraph Partitioning based Approach for Scheduling of Tasks with Batch-shared I/O.* In Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), Vol.2, pp.792–799, 2005.

[60] S. Kim and J.B. Weissman. *A Genetic Algorithm-Based Approach for Scheduling Decomposable Data Grid Applications* Proc. 33rd Intl Conf. Parallel Processing (ICPP04), Vol. 1, pp.406–413, 2004.

[61] A.N. Kolmogorov. *Three Approaches to the Quantitative Definition of Information.* Problemy Peredachy Informatzii, Vol.1, pp.4–7, 1965.

[62] P.R. Krugman. *The Self-organizing Economy.* Blackwell Publishers, ISBN:1557866996, 1996.

[63] C. Lee and M. Hamdi. *Parallel Image Processing Applications on a Network of Workstations.* Parallel Computing, Vol.21, pp.137–160, 1995.

[64] C. Lee anf G. Percivall. *Standard-based Computing Capabilities for Distributed Geospatial Applications.* Computer, Vol.41, No.11, pp.50–57, 2008.

[65] A. Legrand, A. Su, and F. Vivien. *Minimizing the Stretch when Scheduling Flows of Biological Requests.* Research report RR2005-48, Ecole Normale Superieure de Lyon, 2005.

[66] M. Li. and P. Vitany. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, Heidelberg, Second Edition, ISBN:0387339981, 1997.

[67] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R. Freund. *Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems.* Journal of Parallel Distributed Computing, Vol.59, No.2, pp.107–131, 1999.

[68] D.C. Marinescu. *Internet-based Workflow Management: Towards a Semantic Web.* Willey, ISBN:0471439622, 2002.

[69] Matsuoka, K. Saga, and M. Aoyagi. *Coupled-simulation e-Science Support in the NAREGI Grid.* Computer, Vol.41, No.11, pp.42–49, 2008.

[70] J. Machta. *Complexity, Parallel Computation, and Statistical Physics.* Complexity, Vol.11, No.5, pp.46–64, 2006.

[71] D.C. Marinescu, Y. Ji, and G.M. Marinescu. *The Complexity of Scheduling and Coordination in Computational Grids.* In Process Coordination and Ubiquitous Computing, CRC Press, London, pp.119–132, 2002.

[72] D.C. Marinescu, G.M. Marinescu, Y. Ji, L. Bölöni, and H.J. Siegel. *Adhoc Grids: Communication and Computing in a Power-constrained Environment.* Workshop on Energy-Efficient Wireless Communications and Networks 2003 (EWCN 2003), pp.113–122, 2003.

[73] D.C. Marinescu, C. Yu, and G.M. Marinescu. *Self-organizing Sensor Networks.* Proc. Int. Symp. on Wireless Pervasive Computing (ISWPC 2008), CD Proceedings, 2008.

[74] D. C. Marinescu, C. Yu, G. M. Marinescu, J. P. Morrison, and C. Norvik. *A Reputation Algorithm for a Self-organizing System based upon Resource Virtualization.* Proc. 22nd IEEE Int. Parallel and Distributed Processing Symposium (IPDPS08), Heterogeneity in Computing Workshop 2008 (HCW-08), pp.1–6, 2008.

[75] D.C. Marinescu, C. Yu, and G.M. Marinescu. *A Secure Self-organizing Sensor Network.* Proc. Workshop of Pervasive Adaptation (PERADA), pp.114–119, 2008.

[76] D.C. Marinescu, J.P. Morrison, and H.J. Siegel. *Options and Commodity Markets for Computing Resources.* In Market Oriented Grid and Utility Computing, R. Buyya and K. Bubendorf, Eds., Wiley, 2008 (in print).

[77] D.C. Marinescu, J.P. Morrison, C. Yu, C. Norvik, and H.J. Siegel. *A Model for Self-organization of Complex Computing and Communication Systems.* Proc. Second IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems (SASO 08), pp.149–160, 2008.

[78] D.C. Marinescu, C. Yu, and G.M. Marinescu. *Self-organization of Very Large Sensor Networks based on Small-worlds Principles.* In CD proceeding of Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 09), 2009.

[79] W. Matthews and L. Cottrell. *Achieving High Data Throughput in Research Networks.* Technical Report SLAC-PUB-8903, Stanford Linear Accelerator Center, 2001.

[80] M. Mathis, J. Semke, and J. Mahdavi. *The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm.* ACM SIGCOMM Computer Communication Review, Vol.27, No.3, pp.67–82, 1997.

[81] C. von der Marlsburg. *Network Self-organization.* An Introduction to Neural and Electronic Networks, S. Zonetzer, J. L. Davis, and C.Lau (Eds), pp.421–432, Academic Press, San Diego, CA, 1995.

[82] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas. *Data Intensive and Network Aware (DIANA) Grid Scheduling.* J. Grid Computing, Vol.5, pp.43–64, 2007.

[83] M.L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, ISBN:0131655639, 1967.

[84] M.A. Moges and T.G. Robertazzi. *Grid Scheduling Divisible Loads from Multiple Sources via Linear Programming.* IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), pp.423–428, 2004.

[85] J.P. Morrison, S. John, and D.A. Power. *Supporting Native Applications in WebCom-G.* Distributed and Parallel Systems Cluster and Grid Computing Series: The Kluwer International Series in Engineering and Computer Science, Vol.777, pp.147–154, 2004.

[86] S. Olariu and Q. Xu. *Information Assurance in Wireless Sensor Networks.* Proc. Wireless sensor networks and Applications (WSNA'03), Vol.13, pp.236a, CA, 2003.

[87] *OpenPBS.* http://www.pbsgridworks.com/

[88] V. Phua, A. Datta, R. Cardell-Oliver. *A TDMA-based MAC Protocol for Industrial Wireless Sensor Network Applications Using Link State Dependent Scheduling.* IEEE Global Telecommunications Conference (GLOBECOM'06), pp.1–6, 2006.

[89] A. Plastino, C.C. Ribeiro, and N. Rodriguez. *Developing SPMD Applications with Load Balancing.* Parallel Computing, Vol.29, No.6, pp.743–766, 2003.

[90] G. J. Pottie and W. J. Kaiser. *Wireless integrated sensor networks.* Comm. ACM, Vol.43, No.5, pp.51–58, 2000.

[91] G. Nicolis and I. Prigogine. *Exploring Complexity.* Feeman, ISBN:0716718596, 1989.

[92] K. van der Raadt, Y. Yang, and H. Casanova. *APSTDV: Divisible Load Scheduling and Deployment on the Grid.* Technical Report CS2004-0785, Dept. of Computer Science and Engineering, University of California, San Diego, 2004.

[93] C. Raghavendra and S. Singh. *PAMAS: Power Aware Multi Access Protocol with Signaling for Adhoc Networks.* ACM SIGCOMM Comp. Communication Review, Vol.28, No.3, pp.5–26, 1998.

[94] V. Raghunathan, C. Schurgers, S. Park, and M. Srivasatava. *Energy Aware Wireless Microsensor Networks.* IEEE Signal Processing Magazine, Vol.19, No.2, pp.40–50, 2002.

[95] A. Ranganathan and R. H. Campbell. *What is the Complexity of Distributed Systems.* Complexity, Vol.12, No.6, pp.37–45, 2007.

[96] H. Renard, Y. Robert, and F. Vivien. *Static Load-balancing Techniques for Iterative Computations on Heterogeneous Clusters.* Technical Report RR-2003-12, LIP, ENS Lyon, France, 2003.

[97] S. Smallen, H. Casanova, and F. Berman. *Tunable On-line Parallel Tomography.* In Proceedings of SuperComputing' 01, Denver, CO, 2001.

[98] R. G. Smith. *The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver.* IEEE Trans. on Computers, Vol.C-29, No.12, pp.1104–1113, 1980.

[99] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie.“ *Protocols for Self-organisation of a Wireless Sensor Network.* IEEE Personal Communications, Vol.7, No.5, pp.16–27, 2000.

[100] R. Steinmetz and K. Wehrle. *Peer-to-Peer Systems and Applications.* Lecture Notes in Computer Science, Vol.3485, ISBN:354029192X, 2005.

[101] W.R. Stevens. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.*The Internet Society (RFC2001), 1997.

[102] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. *An Economic Paradigm for Query Processing and Data Migration in Mariposa.* Proc. 3rd Int. Conf. on Parallel and Distributed Information Systems, pp.58–67, 1994.

[103] *SunGrid Engine.* http://gridengine.sunsource.net/

[104] S. Singh and C.S. Raghavendra. *PAMAS - Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks.* ACM Computer Communication Review, Vol.28, pp.5–26, 1999.

[105] D. Thain, T. Tannenbaum, and M. Livny. *Condor and the Grid.* In Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003.

[106] H. Topcuouglu, S. Hariri, and M.-Y. Wu. *Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing.* IEEE Trans. Parallel Distrib. Syst., Vol.13, No.3, pp.260–274, 2002.

[107] B. Tsybakov and N. Vvedenskaya. *Random Multiple Access Stack Algorithms.* Prob. Inform. Trans, Vol.16, pp.230–241, 1980.

[108] A. M. Turing. *The Chemical Basis of Morphogenesis.* Philos. Trans. Roy. Soc. London B, Vol.237, pp.37–72, 1952.

[109] S. Viswanathan, B. Veeravalli, and T.G. Robertazzi. *Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems.* IEEE Transactions on Parallel and Distributed Systems, vol.18, pp.1450–1461, 2007.

[110] G. Wang, D. Turgut, L. Bölöni, Y. Ji, and D.C. Marinescu. *Improving Routing Performance through m-limited Forwarding in Power-constrained Wireless Networks.* Journal of Parallel and Distributed Computing (JPDC), Vol.68, pp.501–514, 2008.

[111] G. Wang, D. Turgut, L. Bölöni, and D.C. Marinescu. *Time-parallel Simulation of Wireless Adhoc Networks with Compressed History.* Journal of Parallel and Distributed Computing (JPDC), Vol.69, No.2, pp.168–179, 2008.

[112] G. Wang, D. Turgut, L. Bölöni, and D.C. Marinescu. *A MAC Layer Protocol for Wireless Adhoc Networks with Asymmetric Links.* Ad Hoc Networks Journal, Vol.6, No.3, pp.424–440, 2008.

[113] D. J. Watts and S. H. Strogatz. *Collective-dynamics of Small-world Networks.* Nature, Vol.393, pp.440–442, 1998.

[114] J.B. Weissman. *Prophet: Automated Scheduling of SPMD Programs in Workstation Networks.* Concurrency: Practice and Experience, vol.11, pp.301–321, 1999.

[115] N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam. *TeraGrid Science Gateways and their Impact on Science.* Computer, Vol.41, No.11, pp.32–41, 2008.

[116] W. Willinger, R. Govindan, S. Jamin, V. Paxson, and S. Shenker. *Scaling Phenomena in the Internet: Critically Examining Criticality.* Proc. National Academy of Sciences, pp.2573–2576, 1999.

[117] D. H. Wolpert and W. Macready. *Using Self-dissimilarity to Qunatify Complexity.* Complexity, Vol.12, No.3, pp.77–85, 2007.

[118] R. Wolski, N. Spring, and J. Hayes. *Predicting the CPU Availability of Time-shared Unix Systems.* In Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8), pp.105–112, 1999.

[119] R. Wolski, N.T. Spring, and J. Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.* Future Gener. Comput. Syst. Vol.15, No.5, pp.757–768, 1999.

[120] H.M. Wong, D. Yu, B. Veeravalli, and T.G. Robertazzi. *Data- Intensive Grid Scheduling: Multiple Sources with Capacity Constraints.* Proc. 16th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS03), pp.7–11, 2003.

[121] H.M. Wong, B. Veeravalli, and G. Barlas. *Design and Performance Evaluation of Load Distribution Strategies for Multiple Divisible Loads on Heterogeneous Linear Daisy Chain Networks.* J. Parallel Distributed Computing, Vol.65, No.12, pp.1558–1577, 2005.

[122] A. Woo and D. Culler. *A Transmission Control Scheme for Media Access in Sensor Networks.* Proc. 7th Int. Conf. on Mobile Computing and Networking, pp.221–235, 2001.

[123] A. Woo. *Evaluation of Efficient Link Reliability Estimators for Low Power Wireless Networks.* Technical Report UCB/CSD-03-1270, U.C. Berkeley, 2003.

[124] *Crossbow.* http://www.xbow.com

[125] Y. Yang and H. Casanova. *Multi-round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation.* Technical Report CS2002-0721, Dept. of Computer Science and Engineering, University of California, San Diego, 2002.

[126] W. Ye, J. Heidemann, and D. Estrin. *An Energy Efficient MAC Protocol for Wireless Sensor Networks.* Proc. IEEE Infocom, Vol.3, pp.1567–1576, 2002.

[127] Y. Yu, R. Govindan, and D. Estrin. *Geographical and Energy-aware Routing: a Recursive Data Dissemination Protocol for Wireless Sensor Networks.* Technical Report UCLA-CSD TRE-01-0023, UCLA, 2001.

[128] C. Yu, D.C. Marinescu, H.J. Siegel, and J.P. Morrison. *A Simulation Study of Data Partitioning Algorithms for Multiple Clusters.* 7th IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid 2007), pp.259–266, 2007.

[129] C. Yu, D. C. Marinescu, J. P. Morrison, B. C. Clayton, and D. A. Power. *An Automated Data Processing Pipeline for Virus Structure Determination at High Resolution.* 6th Int. Workshop on High Performance Structural Biology (HiCOMB), pp.1–8, 2007.

[130] C. Yu and D.C. Marinescu. *Load Distribution and Co-termination Scheduling Algorithms for Large-Scale Distributed Applications.* ISCA 21st International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS 2008), pp.117–122, 2008.

[131] C. Yu and D.C. Marinescu. *Divisible Load Co-scheduling for Data-Intensive Applications.* Journal of Grid Computing, 2009. (In Print)

[132] D. Yu and T. Robertazzi. *Divisible Load Scheduling for Grid Computing.* In 15th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS'03), IASTED Press, 2003.

[133] T. Zhu, Y. Wu, and G. Yang. *Scheduling Divisible Loads in the Dynamic Heterogeneous Grid Environment.* In Proceedings of the 1st international conference on Scalable information systems, Vol.152, 2006.