Efficient ressources management in a ditributed computer system, modeled as a dynamic complex system

Mahdi Abed Salman Meslmawy

NORMANDY UNIVERSITY

DOCTORAL SCHOOL SPMII

Efficient Resources Management in a Distributed Computer System, Modeled as a Dynamic Complex System

P H D T H E S I S

to obtain the title of

Doctor of Science

of Le Havre University

Speciality : Computer Sciences

Defended by

Mahdi Abed Salman MESLMAWY

at Le Havre

on October 12, 2015

Members of the jury:

President	Frédéric Guinand	-	Université du Havre
Referees :	Jean Loup Guillaume	-	Université de La Rochelle
	Abderrafiaa Koukam	-	Université de Technologie de Belfort-Montbéliard
Examiner :	Aziz Moukrim	-	Université de Technologie de Compiègne
Advisors :	Cyrille Bertelle	-	Université du Havre
	Eric Sanlaville	-	Université du Havre

Acknowledgments

This thesis was made possible by a French-Iraqi Governments Scholarship. I am grateful to the Both Governments for the scholarship which enabled me to undertake a Ph.D. program in Computer Science at the University of Le Havre.

It is a pleasure to thank many people who made this thesis possible with their encouragement, support and assistance.

I am firstly indebted to my supervisors Professor Cyrille Bertelle and Professor Eric Sanlaville for the continuous support of my Ph.D study and related research, for their patience, motivation, and immense knowledge. It is difficult to overstate my gratitude to them. They provided me with inspiration, invaluable support and advice.

Their guidance helped me in all the time of research during this thesis. I could not have imagined having a better advisors and mentors for my Ph.D. study.

Besides my supervisors, I would like to thank the rest of my thesis committee: Prof. Frédéric Guinand, Prof. Jean Loup Guillaume, Prof. Abderrafiaa Koukam, and Prof. Aziz Moukrim, for their insightful comments and encouragement, but also for the hard question which inspired me to widen my research from various perspectives.

My sincere thanks also go to all members of RI2C who provided me an opportunity to join their team as intern, and who gave access to the laboratory and research facilities. Also, I have not to forget the help provided by the team of the technical support. Without they precious support it would not be possible to conduct this research.

Also, I would like to thank all my colleagues for their help and collaboration. Finally, I offer my regards and blessings to all of those who supported me in any respect during the completion of this work.

Mahdi Abed Salman Meslmawy
Le Havre - France

Contents

1 INTRODUCTION **1**
- 1.1 Motivation . . . 1
- 1.2 Interest of complex system approach . . . 2
- 1.3 Contributions . . . 3
- 1.4 Thesis organization . . . 3

2 PROBLEM DEFINITION AND STATE OF THE ART **5**
- 2.1 Introduction . . . 6
- 2.2 Distributed Computing System (DCS) types . . . 6
  - 2.2.1 Cluster computing . . . 7
  - 2.2.2 Grid computing . . . 8
  - 2.2.3 Cloud computing . . . 9
- 2.3 DCS modeling . . . 10
- 2.4 Modeling of interconnected network . . . 11
  - 2.4.1 Graph theory as modeling tool . . . 11
  - 2.4.2 Types of complex network topology . . . 13
- 2.5 Resources management of a DCS . . . 16
  - 2.5.1 Computing resources . . . 17
  - 2.5.2 Workload Modeling . . . 17
  - 2.5.3 Resource discovery . . . 18
  - 2.5.4 Load balancing . . . 21
  - 2.5.5 Performance evaluation . . . 24
- 2.6 Multi-Agent System as a management tool . . . 24
  - 2.6.1 Agents . . . 25
  - 2.6.2 Environment . . . 25
  - 2.6.3 Simulating a DCS . . . 26
- 2.7 Behavior analysis of a DCS using complex systems . . . 27
  - 2.7.1 Complex system . . . 27
  - 2.7.2 Computational aspects of complex systems . . . 29
  - 2.7.3 Dynamic graph . . . 30
  - 2.7.4 Overlay network . . . 31

3 PROBLEM SOLVING METHODOLOGY **33**
- 3.1 Architectural model of DCS . . . 34
  - 3.1.1 Node description . . . 34
  - 3.1.2 Agent description . . . 35
  - 3.1.3 Network types . . . 37
  - 3.1.4 Workload description . . . 37
- 3.2 Resources discovery . . . 40
  - 3.2.1 Load balancing . . . 40

3.2.2 Improving information management . . . 41
3.2.3 Reinforcement . . . 41
3.2.4 Information preference . . . 41
3.3 Heterogeneous system . . . 42
3.3.1 Nodes with different architectures . . . 42
3.3.2 Nodes of multi-core . . . 43
3.3.3 Communication bandwidth . . . 43
3.4 System state indicators . . . 45
3.5 Interaction capturing model . . . 46

4 THE SIMULATOR 49
4.1 Introduction . . . 49
4.2 Network . . . 49
4.3 Node implementation . . . 50
4.4 Agent implementation . . . 51
4.4.1 Network agents . . . 52
4.4.2 Node agents . . . 57
4.5 Workload implementation . . . 62
4.6 Summary . . . 63

5 RESULTS AND DISCUSSION 65
5.1 Parameters setting . . . 66
5.1.1 Node types . . . 66
5.1.2 Network types . . . 67
5.1.3 Workload types . . . 71
5.2 Simulation results . . . 73
5.2.1 No migration . . . 74
5.2.2 Load balancing with local scheme resource discovery . . . 74
5.2.3 Load balancing with global scheme resource discovery . . . 83
5.2.4 The impact of resource discovery method . . . 93
5.2.5 Structure of the overlay network . . . 94
5.2.6 Improving information management . . . 102
5.2.7 Heterogeneous nodes and bandwidth . . . 105
5.2.8 Analyzing of migration graphs . . . 109
5.3 Discussion . . . 114
5.3.1 Performance of the system . . . 114
5.3.2 Structure of the overlay network . . . 115
5.4 Conclusion . . . 116

6 CONCLUSIONS AND PERSPECTIVES 117
6.1 Conclusions . . . 117
6.2 Perspectives and future work . . . 118

A WORKLOAD TRACES **119**
A.1 Workload Archeives 119
A.1.1 Archive list 119
A.1.2 Log format 119

Bibliography **123**

List of Figures

2.1 Different Graph Models 14
2.2 Ants: a single ant, b a colony. source: [Hoekstra 2010a] 28

3.1 Relationship between local agents 36
3.2 Multi-layered Network 38
3.3 Scheme of a multi-community (architecture) overlay network 44

4.1 Network and node agents communication scheme 52
4.2 Multi-level Agent interaction 63

5.1 Overview of studied elements in a DCS 66
5.2 Inverse cumulative degree distribution of real and generated graphs . 69
5.3 Characteristics of real graphs 69
5.4 Characteristics of generated graphs 69
5.5 Histogram of job arrivals during 1440 cycles 71
5.6 Total MRT for each load balancing strategy on each real graph . . . 76
5.7 MRT for each load balancing strategy for *Identical* job type on real graphs 76
5.8 σQ in generated graphs of $\overline{deg}(G) = 4$ using *Identical* job type . . . 77
5.9 σQ in generated graphs of $\overline{deg}(G) = 4$ using $DiffProcl$ job type . . 77
5.10 σQ in generated graphs of $\overline{deg}(G) = 8$ using *Identical* job type . . . 78
5.11 σQ in generated graphs of $\overline{deg}(G) = 8$ using $DiffProcl$ job type . . 79
5.12 Waiting jobs in generated graphs of $\overline{deg}(G) = 8$ using *Identical* job type 79
5.13 Waiting jobs in generated graphs of $\overline{deg}(G) = 8$ using $DiffProcl$ job type 80
5.14 Total MRT in generated graphs at different $\overline{deg}(G)$, *Identical* job type 80
5.15 Total MRT in generated graphs at different $\overline{deg}(G)$, and $DiffProc$ job type 81
5.16 MRT using two types of distribution of arrival rate in real graphs . . 82
5.17 MRT using two types of distribution of arrival rate in generated graphs 83
5.18 MRT in real graphs using rumor spreading, cache size=4, and *Identical* job type 84
5.19 MRT in real graphs using mobile agent, cache size=4, and *Identical* job type 85
5.20 MRT in real graphs using rumor spreading, cache size=32, and *Identical* job type 85
5.21 MRT in real graphs using mobile agent, cache size=32, and *Identical* job type 86

5.22 Total MRT in real graphs using rumor spreading for using *Identical* job type . . . 86
5.23 Total MRT in real graphs using mobile agent for using *Identical* job type . . . 87
5.24 σQ for of using rumor spreading in generated graphs of $\overline{deg}(G) = 4$, cache size =4, and *Identical* job type. . . . 88
5.25 σQ for using mobile agents in generated graphs of $\overline{deg}(G) = 4$, cache size=4, and *Identical* job type. . . . 88
5.26 waiting jobs using rumor spreading in generated graphs of $\overline{deg}(G) = 4$, cache size =4 . . . 89
5.27 waiting jobs using mobile agents in generated graphs of $\overline{deg}(G) = 4$, cache size =4 . . . 89
5.28 σQ in generated graphs of $\overline{deg}(G) = 4$ for of using: rumor spreading, cache size =8, and *Identical* job mode. . . . 90
5.29 σQ in generated graphs of $\overline{deg}(G) = 4$ for using: mobile agents, cache size=8, and *Identical* job type. . . . 90
5.30 Totoal MRT in generated graphs using rumor spreading with cache size=16 . . . 91
5.31 Total MRT in generated graphs using mobile agent and cache size=16 91
5.32 Total MRT in generated graphs of $\overline{deg}(G) = 4$ using rumor spreading with each cache size . . . 92
5.33 Total MRT generated graphs of $\overline{deg}(G) = 4$ using mobile agent with each cache size . . . 92
5.34 Total MRT on generated graphs using SID with each resource discovery method . . . 93
5.35 Total MRT on generated graphs using RID with each resource discovery method . . . 93
5.36 Total MRT on generated graph using SANDPILE with each resource discovery method . . . 94
5.37 Total MRT on generated graph using HLM with each resource discovery method . . . 94
5.38 In-Degree distribution of overlay networks created on real graphs by rumor spreading method . . . 96
5.39 In-Degree distribution of overlay networks created on real graphs by mobile agent method . . . 96
5.40 In-Degree distribution of snapshots of overlay network for $TTL = 2$ to 5: Left for underlay network of $\overline{deg}(G) = 8$, right for underlay network $\overline{deg}(G) = 48$, Rumor spreading, and $K_D = \infty$ . . . 97
5.41 In-Degree distribution of overlay networks:Left for underlay network of $\overline{deg}(G) = 8$, right for underlay network $\overline{deg}(G) = 48$, Rumor spreading, cache=24 . . . 98
5.42 Features of overlay network on generated graphs of $\overline{deg}(G) = 16$: left from mobile agent, right from rumor spreading. . . . 99
5.43 Total MRT obtained using rumor spreading by each load balancing strategy . . . 100

5.44 Total MRT obtained using mobile agent by each load balancing strategy 101
5.45 MRT resulting from different cooperation schemes in real graphs 1 and 2 . . . 102
5.46 MRT resulting from different cooperation schemes in generated graphs 102
5.47 In-Degree distribution of overlay network resulting from different cooperation schemes, 8B . . . 103
5.48 In-Degree distribution of overlay network resulting from different cooperation schemes, 8R . . . 103
5.49 In-Degree distribution of overlay network resulting from different cooperation schemes, 8E . . . 104
5.50 In-Degree distribution of overlay network resulting from different cooperation schemes, 8W . . . 104
5.51 σQ of heterogeneous type using simple and complex overlay structures 106
5.52 Number of migrated jobs per cycle of heterogeneous type using simple and complex overlay structures . . . 107
5.53 Number of aborted migrations per cycle of heterogeneous type using simple and complex overlay structures . . . 107
5.54 Total MRT of heterogeneous type using simple and complex overlay structures . . . 108
5.55 MRT for node types (H, B) and job types $(Identical, DiffSize)$ . . 109
5.56 Number of arcs in migration graphs resulting from different configurations . . . 110
5.57 Diameter of migration graphs resulting from different configurations 110
5.58 Number of "Sources" in migration graphs resulting from different configurations . . . 111
5.59 Number of "Bridges" of migration graphs resulting from different configurations . . . 111
5.60 Number of "Sinks" of migration graphs resulting from different configurations . . . 112
5.61 Number of "SCC" of migration graphs resulting from different configurations . . . 112
5.62 Size of largest "SCC" of migration graphs resulting from different configurations . . . 113
5.63 Average Clustering Coefficient of migration graphs resulting from different configurations . . . 113

List of Tables

5.1 Node Configuration types 67
5.2 Features of all graph instances 70
5.3 Job types in workload model 72
5.4 Experiments and their configurations 74
5.5 Characteristics of overlay network created by each discovery method over real graphs(out-degree is considered) 95

List of Algorithms

1 Local diffuser 52
2 Exchange diffuser 53
3 Transferor diffuser 53
4 Migrator 54
5 migrate(source,destination1, destination2) 55
6 migrate(source, destination) 55
7 select(destination) 56
8 move(source, destination,job) 56
9 updateIndicator(source,destination) 57
10 Promulgator 58
11 Mobile agent 59
12 checkConstancy(D, preference) 59
13 Scheduler 60
14 balancer 61

Chapter 1

INTRODUCTION

Contents

1.1 Motivation 1
1.2 Interest of complex system approach 2
1.3 Contributions 3
1.4 Thesis organization 3

1.1 Motivation

High-speed computer networks have moved computer system technology into new era. Individual isolated computing units are not enough anymore to handle the rapid increase of demands on more computing power. A high-speed network connects these unites to provide a continuous availability of computing resources.

A distributed computing system (DCS) is a set of computing nodes connected by a communication network. They collaborate by performing tasks for the benefit of each other. Lightly loaded nodes offer doing some tasks for heavily loaded ones to balance load on the system.

Collaboration is a life style of many natural systems. Such system composes of large number of entities. Each entity often does a specific task that is a partial and simple solution of a global and complex one. An entity exchanges interested information with near other entities. These neighbor entities (which know the state of each other) communicate and negotiate to perform the global task. Such systems are called complex systems.

Similarly, processes (called services and located at nodes) of DCS communicate through message passing. If a service controls its behavior without direct intervention from other services, then it is called an agent. A node may host many agents that have been designed to do specific tasks.

An agent is an autonomous entity. It has a working environment. It takes decisions depending on its internal state or/and environments state. It communicates with other local or remote agents to request or report information. An agent changes its state or responds to a request depending on the content of receiving messages [Meshkova 2008].

The global performance of DCS emerges from the interaction between agents. Frequent request-respond communication creates a kind of long-term relations between agents [Bertelle 2007]. A relation may combine more than two agents hosted

in different nodes. A self adapted distributed system is an objective to replace a central managed model of DCS. Self organization appears when communities of nodes evolved from the interactions between agents.

Theories and implementations become developed based on behavior inspired by real life systems. Biologists, physicist, humanists, economists and ecologist are studying models of real life based on self-adapted distributed systems. Each individual of insects chooses its next task depending on little local information. Simple actions of individuals lead to high degree of organization inside the community. Ant Colony algorithm [Blum 2005], [Dorigo 2006] is developed based on the phenomena of self-organization noticed in some type of ant colonies. Swarm intelligent [Bonabeau 1999a], [Lazinica 2009] studies the collective behavior of social communities. It now becomes a field of artificial intelligent science. Literature refers to individuals of these communities as agent.

Natural inspired systems are considered complex systems. In complex systems theory, a system is composed of many subsystems. Relations among subsystems of a complex system and inside subsystems are present. A complex system has a large number of interacting agents without central control [Boccara 2004]. The global behavior of a complex system emerges from the interaction between agents that reside in one or different subsystems. Features of single individual never can express the emergent global behavior.

Monitoring the behavior of a distributed system imposes adding specialized agent at each node to collect information about its activities. To do that with real DCS means adding a huge overhead which is highly avoided. Modeling and simulation is the solution used when it is impossible to monitor each activity of working systems.

In this thesis, a model of a distributed system is built. We use simulation to monitor agents' behavior at global and local levels of system hierarchy. The effect of the network topology on the interaction between agents is computed. The performance of different resources discovery methods in different network topology is compared. An effective age of information and size of agent knowledge are determined. A complex network model is presented to control resources heterogeneity. A new load balancing strategy is introduced. A graph based model to capture interaction between agents is formulated. The role of some parameters is discussed.

1.2 Interest of complex system approach

Many researches of optimizing distributed systems follow an algorithmic approach where the objective is optimizing both the algorithm and/or the parameters. The large number of entities composes a DCS and number of relations between entities makes such method difficult to explain obtained results. Complex system theory provides a new framework that can be used to model DCSs.

In complex systems, the global system behavior is emerged from the local interaction between system entities. A complex system does not impose that an entities

have a complete knowledge about whole system. Usually, interaction takes place between neighbor entities with local knowledge. We aim to build a model of DCS that manages its resource as the complex systems do.

1.3 Contributions

This thesis presents a complex system model in which a dynamic and self-organized overly network is emerged from interaction of system agents. That network facilitates load movement by making extremely varying nodes become neighbors.

The system resources are efficiently managed by redistribute load evenly between system nodes. Load should moved once (or at least few times) from highly loaded nodes to lightly loaded ones.

The original contributions of this thesis are the following: discussing the efficiency of known resource discovery methods, proposing a new load balancing strategy that uses less information, including these into a DCS simulator, developing a self-organization overlay network structure that improves the performance of tested model of DCS, featuring resource discovery heterogeneity, and using the tool of complex system studies to analysis the properties and behavior of the proposed model.

1.4 Thesis organization

The thesis is organized in six chapters. Chapter one (this chapter) declares motivation, goal, and contribution of the thesis. Chapter two gives the reader an overview about the context of the domain. It deals with DCSs models, complex systems, multi-agent systems, and graphs. Chapter three presents a model of distributed computing based on complex system concepts. Chapter four presents the simulation tool that is developed as part of the thesis project to evaluate the model. Chapter five conducts datasets, parameters setting, numerical experiments, and discuses obtained results. Finally, conclusions and perspectives are given in chapter six.

Chapter 2

PROBLEM DEFINITION AND STATE OF THE ART

Contents

2.1 Introduction 6

2.2 Distributed Computing System (DCS) types 6

2.2.1 Cluster computing 7

2.2.2 Grid computing 8

2.2.3 Cloud computing 9

2.3 DCS modeling 10

2.4 Modeling of interconnected network 11

2.4.1 Graph theory as modeling tool 11

2.4.2 Types of complex network topology 13

2.5 Resources management of a DCS 16

2.5.1 Computing resources 17

2.5.2 Workload Modeling 17

2.5.3 Resource discovery 18

2.5.4 Load balancing 21

2.5.5 Performance evaluation 24

2.6 Multi-Agent System as a management tool 24

2.6.1 Agents 25

2.6.2 Environment 25

2.6.3 Simulating a DCS 26

2.7 Behavior analysis of a DCS using complex systems 27

2.7.1 Complex system 27

2.7.2 Computational aspects of complex systems 29

2.7.3 Dynamic graph 30

2.7.4 Overlay network 31

2.1 Introduction

Computing systems differ according to the characteristics of nodes, the topology of communication network, and the resource management method. They are classified according to last factor as central or distributed.

Central computing systems are easy to manage and powerful. They occupy one space (often in one building). Computing nodes are connected by high speed local area network LAN. The coordination of whole works in the system is a responsibility of one node. However, the whole system fails when a central node fails (bottleneck problem).

In distributed computing system (DCS), nodes are autonomous and there is no central control[Coulouris 2011]. Nodes interact, negotiate, and cooperate among them to achieve a global goal. DCS could be described as multi-layer system where many relations exist among entities at same layer and between layers.

A node may represent a server, a cluster, a special sub-network, etc. Nodes communicate through a network. Massage passing is a mechanism of communication used in most DCS. Messages are delivered asynchronously.

The high-speed computer network puts the computing technology into a new era. Development of networking technology progresses rapidly. Many computers become able to collaborate and share their resource for achieving a common goal [Tanenbaum 2006].

Three fundamental characteristics of distributed systems are identified: concurrency of components, lack of a global clock, and independent failure of components [Coulouris 2011].

The global behavior of the system emerges from the interaction between its parts. These interactions (information exchange or job migration) can be captured by building dynamic graphs. The structure of these graphs reveals important global behavior of the system.

The above features are concepts in complex systems theory. Developing and understanding a self-managed DCS is a great challenge to the scientists. In this thesis, we study the behavior of a selected model of DCS under the umbrella of complex systems theory, in order to develop a self-managed DCS and to understand its behavior.

2.2 Distributed Computing System (DCS) types

A DCS can be built from only two computers with free software packages. However, a real production DCS is a large scale system. It has large number of actors. It is out of individual capability to build and manage. DCSs are classified depending on designing objectives like: centrality, performance, number of users, and type of application.

Distributed computing on a DCS is a field of computer science discipline. It includes many topics like networking, scheduling, load balancing, resource discovery, parallel algorithm design, etc. These topics indeed concern subsystems in any

DCS. Most works in this field focus on developing and improving solutions of each subsystem with an attempt to "synthesize" it with other parts. However, we study the emergent behavior of a DCS in an equilibrium state. To have such conditions, a selected system needs to be built, operated, and monitored. We rebuild a framework of DCS from existent policies, strategies, and algorithms that have been developed by other researchers, in addition of some new contributions. Scheduling and parallel algorithm designs are out of the scope of this thesis. In this study, we build a DCS model from networks models, some resource discovery methods, and some load balancing strategies. The bricks on DCS model are a network model, resource discovery model, and load balancing model. Below, brief descriptions of most used DCS types are given.

2.2.1 Cluster computing

A cluster composes of several to tens of high-speed server. Nodes are connected by a high speed local area network LAN. A cluster occupies one space (large hall or building). It interfaces to outside world via a dedicated gateway. From the outside, a cluster is viewed as a single unit. A cluster often has central dispatcher and scheduler. Homogeneity is a characteristic feature of cluster computing.

Clusters widely exist in research or industrial institutes. Some organizations may have more than one cluster coordinated by high speed LAN or fiber optic lines. Usually, each cluster is managed by local IT center.

Depending on the type of application, clusters may serve a number of users that ranges from tens in some specialized computing services to thousands in web hosting services.

An example of a cluster is CRIHAN (for Centre de Ressources Informatiques de Haute-Normandie) [CRIHAN], which is an association created in late 1991 with the support of the Regional Council of Haute-Normandie. CRIHAN objectives are to offer power for application in science (mainly in Physics). CRIHAN has operations around two main axes: computer networks and supercomputing.

In the area of computer networks, CRIHAN is the initiator and the network driver of SYRHANO that interconnects, in very high speed, institutions of the University Normandy, all the schools and colleges, and various public institutions.

Hardware resources: Users of CRIHAN have access to computing resources belonging to two hardware architectures. The principal is a supercomputer named 'Antares' type x86-64 cluster from the IBM iDataPlex series. It has 328 servers (3048 cores) computing, plus 51 servers (624 cores) belonging to the Ecole Centrale de Nantes.

A Power server, Atlas', composed of only four eight-core processors, which is reserved for a small user community. Its processors are few but some codes benefit from its specific architecture.

Software Resources: Supercomputers of CRIHAN have suitable software environment for development and porting parallel architecture codes.

On the other hand, CRIHAN manages on behalf of its users about 35-calculation

software covering different scientific themes and whose licenses are handled case by case: free software, software restricted to a user community, License acquired by its user, or license acquired by the CRIHAN in a mutual context.

Most clusters are centrally managed by central dispatcher and scheduler. However, a cluster is considered a DCS if the nodes (servers) of a cluster are autonomous i.e. provided with their own managing system and can interact with each other.

Clusters rarely operate at their maximum capacity, because they are designed to fulfill the peak demand. Owners of such clusters now offer an access to the resources that are not used most of time. Networked set of autonomous clusters create a DCS under the principles of grid computing. Grid computing provides a kind of scheme that allows restricted access to local resources for remote users [Feldhaus 2012]. In the following, grid computing is presented.

2.2.2 Grid computing

Foster [Foster 2003] defined a Grid as "a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service."

A Computational grid is built through sharing resources from different organizations distributed on a wide geographical area to have a group of users or institutions to collaborate in the form of a virtual organization [Foster 2004].

Resources that belong to different conventional organizations are regrouped under what is called virtual organizations VO's. VO's have no temporal and spatial limitations as conventional organizations do. In VO's, the status of both resources and users are dynamic. No restriction of availability is applied for both resources and users [Feldhaus 2012].

Computational grids are often more cost-effective than supercomputers of equal computing power.

Indeed, Grids are constructed with general-purpose grid middleware software libraries. Many resources providers may use a grid middleware to build grid infrastructure in order to have multiple virtual organizations accessing their computing and storage resources [Foster 2003].

A grid site is a set of resources that is located at one location. A grid site uses grid middleware to offer its resources. Examples of grid middleware are: Advanced Resource Connector (ARC) [NorduGrid 2014], Distributed Interactive Engineering Toolbox (DIET) [DIET], European Middleware Initiative (EMI) [EMI], Globus toolkit [Globus], GridWay [GridWay], and etc.

A grid is considered a DCS, if there is a central dispatching of jobs, but afterwards jobs may migrate with decisions that are taken locally. A central service may be used to manage and monitor the grid sites and virtual organizations. At least four grid services provided by the grid middleware are essential [Feldhaus 2012]: Execution management, Storage management, information management, and security management. Below, they are described briefly.

2.2.2.1 Execution management

These services deal with monitoring and control the workload of computing tasks i.e. processing time of user tasks. Users submit their computing tasks in addition to a description for the requirements to carry out these tasks to a central system called workload management system WMS. WMS is responsible for scheduling user tasks when their requirements match the available resources provided by information management system.

2.2.2.2 Storage management

A set of services with various protocols composing a virtual disk management system is used to retrieve and store data on the grid. One virtual disk may combine many storage capacities from different physical resources and locations.

2.2.2.3 Information management

Information is essential for controlling grid behavior. Information services discover and monitor the status of resource in the grid. Information is stored and gathered onto different levels. It is first collected and stored on grid site levels, and then a copy is sent to the central service provider. This information is used by workload management system to allocate computing tasks to resource when a matching is found. This information may be also used for accounting purpose. Sometimes, historical data is used to optimize the decisions of the workload management system when allocating resource to tasks.

2.2.2.4 Security management

These services restrict access of users to resources. The security system defines access roles in the grid. A role may be used by individual users or virtual organization. Often, these services are managed centrally by a special virtual organization.

2.2.3 Cloud computing

Cloud Computing is a computing system in which all types of resources from physical resources like processors, memory, permanent storage etc. to logical resources like software packages and databases are abstracted as services on demand. Hence, some resources at a computing node are coupled by a logical representation called virtual machine (VM). A large computing center (often several clusters) hosts a large number of VMs depending on the amount and types of resources that it may wish to provide. VM can be migrated from one physical machine to another one to have a better performance. Virtualization allows resource providers to safely and effectively provide their services to users.

In cloud computing, client access the computing resources (hardware and software) as a service over a network (typically the Internet).

Two networks often concern the cloud computing architecture [Borko Furht 2010]: The network that allows the end users demanding cloud services to access the cloud data centers. In addition, especially for large-scale cloud data centers, a network connects many specialized computing nodes creating a single-purpose cloud service data center. These nodes are managed centrally by a dedicated IT center. However, a decentralized fashion may be applicable also.

Cloud computing allows companies to avoid building and maintenance costs of the IT centers, and focus on projects by which their businesses are attributed instead of the infrastructure. In addition, it allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and enables IT to adjust more rapidly resources to meet fluctuating and unpredictable business demand. *Google* and *Amazon* are examples for cloud service providers.

A web browser or a light-weight (desktop or mobile) application is designed for end users to access cloud-based services at a remote location where the service managing software and user's data are stored [Miller 2008].

Users can benefit from the competition among resource providers for having service at reasonable cost.

In this thesis, a DCS is any connected set of nodes where each node is a representation for an individual server, a cluster, a grid site, or in general a computing unit that can process user jobs. Nodes communicate over a dedicated network or the Internet.

2.3 DCS modeling

The model is an abstracted and simplified description for the relevant entities of the system design. Depending on abstraction level, three types of DCS models are often defined [Coulouris 2011]:

1. Physical models: explicit description for the hardware composition of a system on the hardware level by terms of computers and other devices and their interconnecting network.

2. Architecture models: functioning description of the main components of the system and the way by which they interact (software architecture), and their mapping on an underlying network of computers (system architecture).

3. Fundamental models: description of the attributes that are formally defined for architecture models. Three fundamental models are defined: interaction models, failure models and security models. Interaction model studies the structure and sequencing of the events between system entities. Failure model studies the ability of the system to recover from a failure. Security models evaluate the degree of data safety and the protection against the unauthorized access.

In this thesis, we concentrate the focus on the modeling of networks, nodes, and the interaction among nodes over a given network. In the following, we present

descriptions for networks modeling tool, nodes' resources and workload, and the types of interaction including resource discovery and load balancing.

2.4 Modeling of interconnected network

The nodes of DCS communicate by exchanging messages over links that are connecting them. A message is a data segment which is often a part of a dialogue between the sender and the receiver. A link is a set of transmission media (such as cable, fiber-optic and wireless channels) and hardware devices (such as routers, switches, bridges, hubs, repeaters and network interfaces). When all the parts of the links are considered then we refer to a physical link while a logical a link is considered when hardware details are not included in the description.

A non-exhaustive list of systems that take form of network includes Internet, social networks, neurons system, roads network, etc [Newman 2003]. For analysis and development purposes, a network is often modeled mathematically using graph theory. Below we review this tool.

2.4.1 Graph theory as modeling tool

Graph theory is the most important mathematical technique used to model the geomorphological relations among the entities in a problem domain. Nodes represent entities. Links connect nodes to show an existent relation between them.

The first research of graph theory was carried by Leonhard Euler on the Seven Bridges of Königsberg [Biggs 1986][1]

The graph theory has application to many domains like: anthropology, biology, chemistry, computer science, geography, linguistics, music, physics, political science, psychology, social science [Wilson 1970][Van Der Hofstad 2009][Gross 2013].

In computer science, graphs are used to represent networks of communication, the flow of computation, etc. For instance, the link structure of a DCS can be represented by a graph, in which the vertices represent computing nodes and edges represent communication between two nodes.

A graph is denoted by $G = (V, E)$ where V is the set of vertices and E is the set of links(edges). Two vertices $u, v \in V$ are neighbors if and only if $(u, v) \in E$.

A directed graph (called digraph) is denoted by $G = (V, A)$, where A is the set of arcs. For arc (i, j), i and j are its source and target respectively. i is called predecessor of j and j is the ancestor of i.

Many statistical measures and indices are used to describe a graph structure. Bellow, definitions to some measures and indices used to characterize a graph are given.

[1] The city of Königsberg in Prussia (now Kaliningrad, Russia) was divided by the Pregel River into four regions including two islands. Regions were connected to each other by seven bridges. The problem was how to walk through the city and cross each bridge once and only once. However, the islands are reachable only by the bridges and every bridge should be used once. Lacking a mathematical technique to analyze and test proposed solutions was the main challenge [Biggs 1986].

- Order $N = |V|$: refers to the number of vertices (nodes) that compose a graph.
- Edge (or arc) count $|E|$ or $|A|$: is the number of existing edges (arcs) in a graph.
- degree $deg(v)$: is the number of edges connected to vertex v.
- In-degree $deg^-(v)$: is the number of arcs entering to vertex v in a directed graph.
- Out-degree $deg^+(v)$: is the number of arcs leaving to vertex v in a directed graph. A vertex with $deg^-(v) = 0$ is called a *source*, as it is the origin of each of its incident edges. Similarly, a vertex with $deg^+(v) = 0$ is called a *sink*.
- average degree: is the average number of edges per node $\overline{deg}(G) = \frac{2|E|}{|V|}$ for an undirected graph, and $\frac{|A|}{|V|}$ for both $\overline{deg^-}(G)$ and $\overline{deg^+}(G)$ in a directed graph.
- Degree distribution: the function (histogram) of how a given degree is frequent in the graph i.e. $V(k) = |\{v : v \in V \ , \ deg(v) = k\}|$, where k is the given degree. A probability density function $pdf(k) = \frac{V(k)}{|V|}$ is considered when the frequency is replaced by the fraction of nodes having a given degree. Moreover, an Inverse Cumulative Degree Distribution ICDD is frequently used in literature for two reasons: it produces a more readable figure for the graph degree distribution and it is easily to fit to a standard distribution like Poisson or power-law. ICDD is computed as $ICDD(k) = 1 - \sum_{j=1}^{k} pdf(j)$.
- Density: the percent of number of edges to total possible number of edges in a graph i.e. $\frac{2*|E|}{(|V|*(|V|1))}$ for undirected graph and $\frac{|A|}{(|V|*(|V|1))}$ for directed graph.
- Path: is an ordered list of edges/arcs that have to be traversed to get from one vertex called the *origin* to another called the *destination*.
- Shortest path: is the path of the minimum distance between two vertices in a graph.
- Diameter $d(G)$: is the longest shortest path in a graph.
- Clustering coefficient of the vertex v is the density of the sub-graph that is composed only from v's neighbors.
- Average clustering coefficient $\overline{C}(G)$of a graph is the mean value of the clustering coefficient of all vertices in a graph.
- connected component: A component is a set of nodes (graph or sub-graph) and it is called connected when there exists a path (without considering the direction of the edges) between any two nodes belong to that set.

- In a directed graph, a component is called weakly connected if a connected (undirected) graph is resulting from replacing all of its directed edges with undirected ones.
- A component is strongly connected SCC if there is a path in each direction between each pair of nodes of the component.

Researchers model Internet or the network of a DCS by a graph. Hence, vertices represent computing nodes. A node has several properties like unique identifier, computing power, connection bandwidth, etc.

A Link (edge or arc) is a representation to a path between two nodes. A path is a series of hopes that a message has to traverse to reach the destination node.

2.4.2 Types of complex network topology

Types are proposed according to specific applications. Selected types that are suited for DCSs are reviewed in the following.

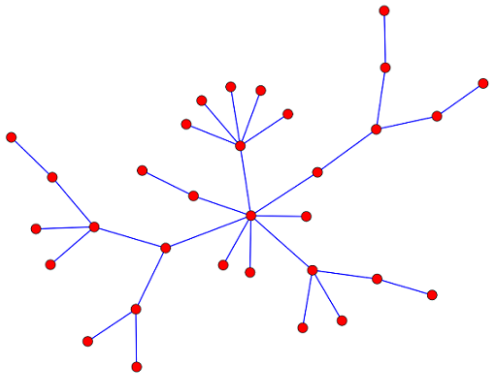
2.4.2.1 Barabàsi Albert

The Barabási-Albert model [Barabási 1999] creates random scale-free networks using a preferential attachment mechanism. Many observations for Scale-free networks are found in natural and human-made systems, like the Internet, the World Wide Web, articles citations, and some social networks. Scale Free Network is defined by a unique feature: power law distribution for the degrees. Barabási-Albert networks are generated using two important concepts: growth and preferential attachment. Both concepts exist widely in real networks. Growth states that the number of nodes in the network increases over time. Preferential attachment means new nodes prefer to attach themselves to highly connected older nodes in the network. Hence, the probability of selecting a particular node to attach to it is proportional to its degree. The number of edges by which a node connects itself to existing nodes for the first time is a parameter to the algorithm. It is either considered a constant or it is chosen at random $\sim [\frac{e}{2}, e + \frac{e}{2}]$, where e is the number of edges parameter. Hence, the average degree of the resulting graph is equal to $deg(G) = e$. In latter case, the minimum edge is equal to $\frac{e}{2}$, see 5.1.2.

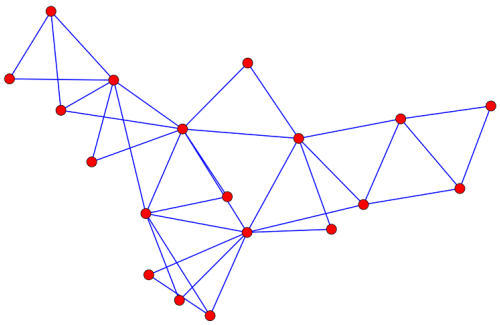
Figure 2.1a shows a graph generated according to Barabási Albert model. It contains 32 nodes and 31 edges. It has a diameter of eight edges. The maximum node degree is eight, there are of 19 nodes with a degree equals to one.

2.4.2.2 Dorogovtsev Mendes

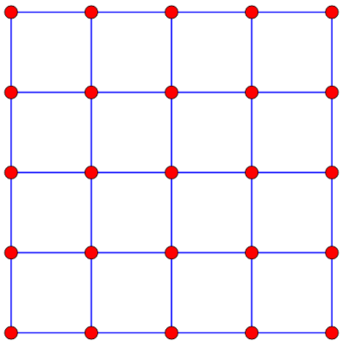
The Dorogovtsev and Mendes[Dorogovtsev 2002] algorithm creates two connected nodes. Then add one node at a time. Each time a node is added, it makes a triangle with a randomly chosen edge. This process has a power-law degree distribution, as nodes that have more edges have more chances to add extra edges since its edges have higher probability to be selected. The Dorogovtsev - Mendes algorithm



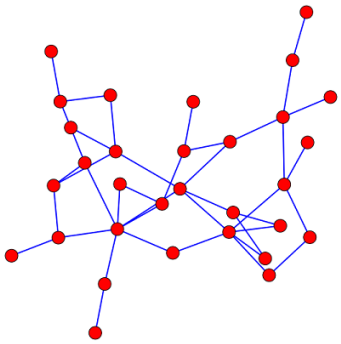
(a) Barabàsi Albert



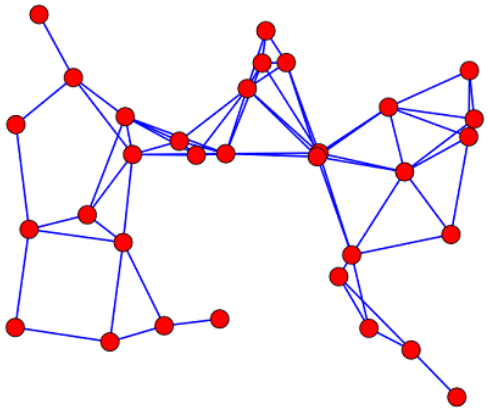
(b) Dorogovtsev Mendes



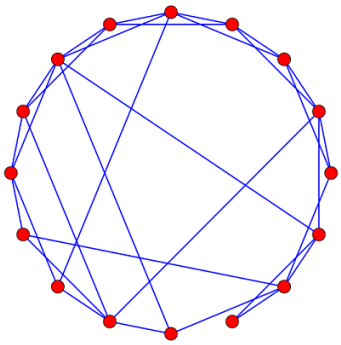
(c) Grid



(d) Random



(e) Random Euclidean



(f) Watts-Strogatz

Figure 2.1: Different Graph Models

always produces planar graphs. Planner graph is that could be drawn on a one dimensional plan. This graph type is not used in our model but it is presented as another instance of a graph with power-law degree distribution.

Figure 2.1b shows a graph generated according to Dorogovtsev and Mendes

algorithm. It contains 19 nodes and 35 edges. It has a diameter of five edges. The maximum node degree is 8; there are of 7 nodes with a degree equals to two.

2.4.2.3 Complete Graph

A complete graph is a graph in which every pair of distinct vertices is connected by a unique edge in case of undirected and two arcs for each direction in case of directed graph. Edges count is $\frac{N(N-1)}{2}$. If it was directed the number of arcs will be $N(N-1)$, where N is number of nodes.

2.4.2.4 Grid graph

Grid is a graph of two dimensions (M x N) where each node is (except of rows 0,N-1 and columns 0,M-1) connected with its four neighbor, or three dimensions where each node is connected with its eight neighbors. When $N = M$ is taken, a square grid (or lattice) is considered. For example an infinite lattice is used in [Bak 1987] and [Laredo 2014] to perform the simulation of the Sandpile model, see 2.5.4. If the grid is wrapped from its ends, a torus is produced. Torus is regular graphs with all nodes have exactly the same degree. Figure 2.1c shows a grid graph where the diameter is eight edges, nodes count 25, edges count 40, Maximum node degree is four, and minimum node degree is two.

2.4.2.5 Random graph

A random graph is generated by starting with a set of N isolated nodes and adding successive edges between them at random. Different models introduced according to definition of randomness property. For Example: Edgar Gilbert model [Gilbert 1959], $G(n, p)$, imposes that a graph G is generated by adding each edge with independent probability $0 < p < 1$. In Erdös, P. Rŕnyi model [Erdös 1959], a graph with N nodes and M edges is chosen uniformly at random from all possible graphs that can be generated using these two components. A random graph is characterized by short diameter but small average clustering coefficient. Random graphs are a general model that can be used as a reference for most real network types. Furthermore, a range of complex networks share features of random graphs [Newman 2005]. Figure 2.1d show a random graph generated for $G(31, 0.2)$ that contains 40 edges, the diameter is 8 edges, maximum node degree is 7, and minimum node degree is one.

2.4.2.6 Random Euclidean

This Algorithm creates random graphs of any size. Cartesian coordinates of nodes are assigned randomly. Hence, each node should have a position attributes vector. A link is created between any two nodes if the Euclidean distance between two nodes is less than a given threshold. Random Euclidean graphs have a relatively long diameter but a high clustering coefficient. They are used to model network

types when node attributes include spatial information, as in the modeling of ad-hoc wireless networks [Nekovee 2007].

Figure 2.1e shows a Random Euclidean graph: diameter is 13, average clustering coefficient is 0.63, nodes count 32, edges count 65, maximum node degree is 7, and minimum node degree is 1.

2.4.2.7 Watts-Strogatz

The Watts-Strogatz model generates a random graph with small-world properties, like short average path lengths and high clustering coefficient. Small world means that most nodes are not adjacent of one another, but the distance between any two randomly chosen nodes is proportional to the logarithm of graph order. It was proposed by Duncan J. Watts and Steven Strogatz in their joint 1998 Nature paper [Watts 1998].

Given graph order N, an even integer K as the mean degree so that $N \gg K \gg \ln(N) \gg 1$, and parameter β, in $(0, 1)$. The algorithm generates an undirected graph in two phases:

- Construct a regular ring lattice, a graph with N nodes each connected to K neighbors, $K/2$ on each side. That is, if nodes are labeled $n_0 ... n_{N-1}$, there is an edge (n_i, n_j) if and only if $0 <\mid i - j \mid mod\big(n - \frac{K}{2}\big) \leq \frac{K}{2}$.
- For every node $n_i = n_0, \ldots, n_{N-1}$ take every edge (n_i, n_j) with $i < j$, and according to probability β, replace (n_i, n_j) with (n_i, n_k) where k is chosen with uniform probability from all possible values that avoid self-loops and link duplication.

A graph is considered small-world, if its average clustering coefficient $\overline{C}$ is significantly higher than a random graph of the same vertex set and diameter length.

Figure 2.1f shows a small-world graph: diameter is 4, Density is 0.25, Average clustering coefficient is 0.269, nodes count 16, edges count 32, maximum node degree is 6, and Minimum node degree is 2.

The information network web, and other real networks have that small world feature [Watts 1998] [Newman 2003] [Little 2007]. Watts-Strogatz, Random, and scale-free models are used in [Fukuda 2007a] to study the impact of network structure on the performance of load balancing.

2.5 Resources management of a DCS

The operating cost of DCSs does not affect (with an exception for energy) by the amount of computation made. Hence, using idle computing resources is the main objective of building a DCS. Management of (especially for large) DCS resources is a non-trivial task. The management process includes two interdependent processes: discovery and balancing. Monitoring imposes known of both resources state and workload state. Balancing uses monitoring information to redistributes workload

i.e. changing state of resources and workload. Below, a description for computing resources and workload characteristics are given followed by details of load balancing and resource discovery processes.

2.5.1 Computing resources

Resource is a helper entity where a profit is gained from its usage. Resources either physical like material or logical like service. Typically, finite resources are used with competition among consumers when the rate of available amount to requested amount is less than one. A controlled usage for such types of resources is a challenge. Computing resources include both physical like: processor, memory, permanent storage, etc and logical like operating system, software packages, compilers, etc. Computing resources are often limited and their management is an objective for both commercial companies and scientific institutes.

Resources in DCS are an aggregation of set of local resources that belong to individual nodes (computer systems). They can be accessed from other computers by means of communication. Each resource is often managed by a program (called service) that provides an interface controlling access and usage of the resources [Coulouris 2011].

Typically, computing resources of most computer systems are listed in the following:

- Central Processing Unit CPU (or a processor) an integrated circuit designed to execute written programs.
- Random Access Memory RAM (or just memory) is a temporary storage used to load programs and data before executions. Typically, RAM or memory in general has its own memory management system MMS. MMS interface between processor and the peripherals such as disk, network adapters etc. RAM lost its content when lost power current.
- permanent storage is used to store programs and data for long duration. It characterized by large capacity and slow access speed as comparison with RAM.
- network connection the simplest one is used to peering two computing devices. It consists of network adapter and connection media. Network equipments such as switches, routers are used in large networks.
- display processor or graphics processing unit GPU is a device attached to display adapters that used by some application which need a special featured-integrated circuit to manipulate visualization processes.

2.5.2 Workload Modeling

DCSs are built to process a large set of user jobs (the workload). The Workload size and job characteristics determine the behavior of a given DCS. Many traces of

working production systems are used in the literature to study, analyze, or evaluate the performance of the developed algorithm. Most traces are archived and placed available for download, for examples: Parallel Workload Archive [PWA] and Grid Workload Archive [GWA]. Traces are formatted in Standard Workload Format (.SWF). This format stores each workload item (job) in a line in a text file. A line contains several fields that are related to one job. The description of SWF format is listed in Parallel Workloads Archive web page [PWA], a copy to the content of this web page is included in A.1 .

Though, traces can be used directly in simulation models, they lack of generality [Lublin 2003]. A trace represents a particular configuration of source system. It may have error or missing data. Instead, synthetic workload generation models are built based on statistical distributions of real workloads. The output of these models is often used to evaluate model behavior in simulated environments [Li 2009].

Production systems vary in the amount and the type of workload they serve depending on the nature of system users. However, most systems have similar (to some degree) statistical patterns. In [Lublin 2003] [Li 2005], authors showed that workload of several production system has yearly, weekly, and daily cycles in its distribution. In a yearly cycle, some particular months have workload over the average. In a weekly cycle, arrival rate pattern is approximately same for all clusters along with the trace duration. In daily cycle, the most important, workload has same distribution during weekdays.

The interarrival time and the service time are the most important attributes that should be considered when generating workload instances. Interarrival time is sampled from a simple distribution e.g. Poisson [Lemoine 1989] or a multi-phase distributions e.g. a two-Gamma distribution [Lublin 2003].

However, other attributes including jobs size, number of requested CPU, requested memory, etc are also have their importance when a model is built to evaluate a related impact. More details are given in 3.1.4.

When job arrival rate varies from node to node, the resource discovery is mandatory for moving load among nodes. Below, this type of interaction between nodes is presented.

2.5.3 Resource discovery

Resource discovery includes two phase: searching (or advertising) and describing. In DCSs, the state of resource is dynamic. At any time, without any notification, new resources may enter the system and existing resources may leave or become unavailable. In such cases, local scheduler needs to know about the state of existing resources to do its task correctly.

Because of dynamic state of resources, information is updated frequently. Information is collected through frequent communication between nodes. This, of course, adds more computing and communication overhead to the system but it is indispensable.

Resource discovery problem is widely studied since 1999 with the research of

Harchol-Balter, Leighton and Lewint in [Harchol-Balter 1999].

Most developed algorithms are based on the communication between neighbor nodes. Algorithms differed in three aspects: type of information to deliver (current node state or all its knowledge), the number of neighbors to inform, and when to inform a neighbor.

Mor Harchol-Balter et. al. [Harchol-Balter 1999] propose several natural and simple distributed algorithms to solve the problem of resource discovery in distributed networks. All of their algorithms involved nodes asking to one or more neighboring nodes for information. A node responds with whole knowledge, or part of it knowledge to an asking node. Their analysis showed that the Name-Dropper algorithm achieves better performance both with respect to time complexity and with respect to the network communication complexity. In Name-Dropper algorithm, each node informs a randomly chosen neighbor node about all known nodes. The receiver node updates its knowledge. The time complexity was computed as the number of time units required to inform each node about all other nodes.

Volgaris et al. [Voulgaris 2005] presented the Newscast model, which is "an epidemic protocol for disseminating information in large, dynamically changing sets of autonomous agents". Authors showed that the snapshots of an overlay network (they called it series of the communication graphs) of this model exhibit stable small-world properties. These properties are not intended or expressed explicitly by agents design, but they are emergent from the underlying simple epidemic-style information exchange protocol.

A modeling framework to Peer-to-peer Data Networks PDN based on complex system theory has been introduced and applied by Shahabi C and Banaei-Kashani F [Shahabi 2005]. The work focused on the efficiency of search in PDNs. Authors proposed a distributed index structure, called the SWAM access method, to have high degree matching of different search queries in indexed PDNs. SWAM is inspired by the "small-world" models originally introduced to explain efficient communication in social networks. The performance of these search mechanisms is evaluated using analytic and numerical methods.

Forestiero [Forestiero 2007] showed that the ANT-based Algorithm for RE-Source management in Grids (Antares) algorithm that can replicate and sort information descriptors on the Grid in regions where it frequently demanded. The algorithm is inspired by the behavior of natural ants [Dorigo 2006]. Authors developed their algorithm by associating a key that represents spacial information of its region to the information descriptor (information vector). Then, these descriptors are sorted and each node collects descriptors that are most similar to its position. In this work, information is collected by a sophisticated resource discovery method and sorting is made according to a dynamic key that represents the preference of the node.

We say a local scheme is used when a node always knows the state of its direct neighbors and only them. Global scheme are used when node suppose to know about nodes other than its direct neighbors. The number of hops a piece of information is allowed to traverse determines the scope of knowledge. In addition to local

scheme, two sophisticated resource discovery methods are selected in this work for comparing system performance in different underlay network structures. They are mobile agent and rumor spreading methods. In the following, we will review briefly each one.

2.5.3.1 Mobile agent

Mobile agent [Braun 2005] is a programming object encapsulated as code and data. It is capable of :(1) traveling between computing nodes, (2) working for another agent, (3) listening to external changes through environment state variables and (4) controlling its internal working state to meet its objectives. These abilities give an agent some artificial intelligence features.

The mobile agents either go back to the home node to report the result or terminate after completion of their assigned mission [Meshkova 2008]. Mobile agents are autonomous. They always carry the required (a small) data with them. Nodes that were unknown when mobile agent was originally initiated can be visited. At each new node, it can make decisions based on its history of visited nodes and the current one. One of the important roles of mobile agent is that information is being disseminated at every node that it visits. A mobile agent based solution is very fault tolerant. Even if some of the mobile agents are terminated, all surviving ones will have a positive impact [Dunne 2001, Charpentier 2005].

2.5.3.2 Rumor spreading

Rumor spreading is one of gossip-based algorithms that attract high attention in last years. They are using simple, efficient, and robust protocols to diffuse information in large networks [Giakkoupis 2011, Haeupler 2012]. In rumor spreading algorithm, an information message called "rumor" at one node (denoted by start node) should reach all other nodes in iterative manner. In each step informed nodes (who become know the rumor), have to tell it to one of its neighbors. A neighbor is chosen uniformly at random. That is referred to as PUSH protocol. In Pull protocol, uninformed nodes(who do not yet know the rumor) ask one of its neighbors to tell it the rumor. The two protocols are also used in one schema called PUSH-PULL protocol where in each step a node either tells or asks the rumor to/from one of its neighbors. All mentioned protocols of this algorithm were proposed in [Demers 1987] in the subject of consistency of replicated distributed databases systems as effect writing queries.

The running time (required to inform all nodes) of a rumor is widely studied, for examples [Chierichetti 2010], [Fountoulakis 2010], [Giakkoupis 2011], Haeupler2012, [Clementi 2013], [Giakkoupis 2014], etc. The objective is computing the number of cycles need for informing a message from a node to all other nodes in the network. In [Censor Hillel 2010], partial information spreading was considered where each node had to receive a specific number of messages from other nodes. This relaxation is practical in a Multi-Agent System models, see 2.6, where each

agent knows about the state of a small set of agents (nodes) called neighbors. However, a little work is made to consider the impact of using delivered information. We study the performance of load balancing strategies that is carried locally by each node depending on a small set of information, which is delivered by local scheme, mobile agent, or rumor spreading method.

2.5.4 Load balancing

Computing nodes collaborate to achieve a global objective. Load balancing is a method by which load is distributed evenly among system nodes. The process is managed in a decentralized way, where a node adapts its load level with its local neighbors and gradually the global system load is balanced.

H. Willebeek-LeMair [Willebeek-LeMair 1993] proposed five load balancing schemes. However only Receiver Initiated Diffusion RID and Sender Initiated Diffusion SID schemes are using local knowledge that is global load balance is obtained through local knowledge and decisions. In SID, nodes frequently broadcast their current load status to all of their direct neighbors. A node initiates a migration towards another node whose load is below a threshold. In RID, a node whose load drops below a threshold T_{low}, requests an overloaded nodes whose load is greater than a threshold T_{high}. We compare the performance of both SID and RID with a method proposed in this thesis.

Siu-Cheung Chau and Ada Wai-Chee Fu [Chau 2003] presented an improved dimension exchange method(DEM) for synchronous load balancing for hypercube architecture. Authors addressed the problem of load imbalance in computing clusters between several universities in Canada. They assumed that the job schedulers of each computing cluster are connected by a hypercube network. By this assumption, they could group clusters that are connected directly through high-speed link in the same sub cube to take advantage of their connections. Furthermore, they could also make use of the topology of a hypercube to reduce the amount of communications during load balancing.

Cao et al. [Cao 2003] presented a load-balancing framework called Mobile Agent based LoaD balancing (MALD) that uses mobile agents to achieve load balancing on distributed web servers. A server management agent (SMA) which is a stationary agent is responsible for monitoring the workload on a local server. Using sender initiated policy, when a server is overloaded; its SMA initiates the load reallocation process. The SMA selects the jobs from local job queue and dispatches the job to other servers.

In Condor [Thain 2005], user sends jobs to an *agent*. The Agent keep the job underhand while searching for *resource* compatible to run the job. Agent and resource advertise themselves to *matchmaker*. Matchmaker matches between the agent and resource and sends the result to the agent. Agent then contacts a resource to revalidate the match. They have to create two process one at each side. Agent creates a *shadow* as front end of execution process. The resource create a *sandbox* which is the execution environment at back end.

Paul Werstien and et al. [Werstein 2006] proposed a dynamic load balancing algorithm. Authors considered CPU queue length, network bandwidth and memory utilization as load metric. The experimentation results with proposed algorithm had shown better performance than traditional algorithm that considers only CPU queue length as metric.

K. Fukuda et al. [Fukuda 2007a] studied the impact of network model on the performance of load balancing. They analyzed on the effectiveness of using statistical properties of the network structure in multi-agent systems. In their work, two types of networks are used: Internet topology and networks generated using theoretical models. An Internet topology is provided by CAIDA. Two theoretical networks are also used: generalized Barabázi-Albert (GBA) model and connecting nearest-neighbor (CNN) model. Authors dealt with optimizing server deployment and client selection algorithms in Internet. An evaluation has been made for two types of server agent deployment algorithms: degree-oriented and random. In degree-oriented deployment algorithm, n agent servers are deployed in the top n nodes that have highest degree i.e. agent servers occupy hub-nodes. In random deployment algorithm, server agents are deployed randomly for nodes with small degree.

Other non-occupied nodes are assigned to client agents. Either a client agent selects its server randomly or through using an algorithm called reverse-waited degree (RWD) selection algorithm. The distance is minimized measured for both number of hops and server load that is estimated by the degree of the server agent node. A scope parameter r is used to limit the length of path (number of hops) that is reachable for a client agent. Hence, when $r = 0$ means interaction is allowed with the node only, $r = 1$ only direct neighbors can interact, etc.

Simulation result for random network showed the performance did not affected when using whatever deployment algorithm. Even when proposed selection algorithm is used, no enhancement is noticed. The degree distribution of random network is blamed here.

For scale-free network, many direct neighbor client agents know a server agent. Hence, the load is initially distributed. The results are good for $r = 1$. For other values of r, result does not show up an enhancement.

Other result showed that the scale-free characteristics and negative degree correlation of a network play an essential role in the performance of selection algorithm.

we investigate the impact of both static and dynamic structure of the overlay network on the performance of several load balancing strategies. Many real in addition to theoretical network models are considered. Then, we introduce three different mechanisms for building a self-organized overlay network that emerged from the interaction among agents. The emerged structure of the overlay network shows a positive impact in comparison with default version even for small world-network.

J.L.J Laredo et. al. [Laredo 2014] presented an on-line and decentralized scheduler based on the concept of Self-organized Criticality SOC model classically called sandpile. The concept was introduced by Back et al. [Bak 1987]. The sandpile is

formed by dropping grains of sand slowly on a small area of a plane. First, grains are distributed arbitrary. Then, gradually a filled cone (pile) is formed when grains avalanche from its top and surface down until the diameter of the base circle.

The hypothesis of this work considered the emergent behavior of SOC is applicable for the problem of dynamic load balancing in large-scale distributed system. The analog is made between tasks and grains, and between avalanche and reallocation of tasks. The work also studies the impact of network topology on the model performance. The workload introduced in form of Bags-of-Tasks (BoTs). First, all jobs were considered present from the first moment. A constant arrival rate also had been tested.

Authors proposed three modifications on previous works to enhance obtained results: First, a new rule to trigger an avalanche. Instead of using fixed threshold for difference between two neighbors piles, a pile avalanches when its height is larger than the sum of two neighbors. Second, a small world network rather than a lattice 2D grid connect the nodes. Moreover, the third, a gossip-based version computes virtual avalanche to find the final destination to receive migrated job instead of multiple migrations.

Authors showed through simulation that small-world topology together with gossip-based virtual avalanche leads a result near the optimal.

Typically, load balancing has four main policies [Cao 2003][Willebeek-LeMair 1993] listed below:

1. Information: the way by which information is provided to the load balancing process manager (section 3.2 gives some selected methods). No information means, load balancing is impossible or uncontrolled.

2. Initiation: When imbalance is detected, who should trigger the load migration process?

3. Source and destination: when the decision is taken to move some load, characteristics of the source (among overloaded nodes that have load more than the average load of set of known nodes) and the destination (among underloaded ones that have load less than the average load of set of known nodes) should be specified.

4. Load selection: determines the properties of the load that is more suitable to be migrated to the destination node and the profitability gained from this migration.

Each policy may have many options, which results in many applicable strategies. Selected options affect the performance of load balancing (consequently, the behavior of the global system is affected). Our aim is to study the global behavior for different initial parameters. The load balancing is a good example of collective collaboration between system entities.

Four strategies are tested in this thesis. They differ either in policy 2, 3 or both. The other policies are same for all selected strategies. Policy 1 tests local

scheme, mobile agent, and rumor spreading. Policy 4 selects for migration the job of latest arrival time or the job of smallest size when nodes differ in their connection bandwidths.

Note that in these descriptions, the neighborhood of a node i is the set of nodes whose load is known by i, and cycle is the time unit.

1. SID: Any overloaded node initiates the migration process [Willebeek-LeMair 1993]. It chooses randomly one underloaded node as destination from its neighborhood. Hence, each cycle, an overloaded node can send only one job but an underloaded one may receive several jobs from different nodes.

2. RID: An underloaded node looks in its neighborhood for an overloaded node to migrate loads from it [Willebeek-LeMair 1993]. A possible source node is chosen at random. Hence, each cycle the initiator can receive only one job while the sender may send several jobs to different nodes.

3. SandPile: The load of a given node is avalanched (dropped) down to some neighbor nodes, if some criteria are met. For example in [Laredo 2014], a node chooses two neighbor nodes at random. Load is distributed evenly among the three nodes when the load of current node is greater than the summation of the two. Hence, an overloaded node may send several jobs to its two neighbors. One migration in sandpile may trigger other migrations in a cascading way until no migration is possible any more. Hence, the network should reach an equilibrium state each cycle. In this work, this constrain is restricted by inspecting each node only once per cycle.

4. HLM [Salman 2014]: This strategy is RID, except that an underloaded node demands a migration from the maximum loaded node in its neighborhood. Hence, a heavily loaded node may respond to many requests of migrations during one cycle.

2.5.5 Performance evaluation

A model of DCS is sensitive to initial values of the parameters. The entire system is in equilibrium state if all nodes have the similar load level. The deviation in queue length is used to measure the degree of balancing between system nodes at any time.

A steady state of the system is achieved when the value of given measure (queue length for example) become constant. The system is going to crash when the queue length keeps increasing.

2.6 Multi-Agent System as a management tool

A Multi-Agent System (MAS) is a set of software agents each with limited ability and partial knowledge about their working environment. An agent affects and feels

the changes in its working environment or at least in its "range of view". Agents cooperate to solve a global task that cannot be solved individually [Vidal 2007].

MAS spread information on entities through neighbor diffusing and not central points. Agents search, learn, and find final solutions. Multi-agent system is composed of agents and their environment.

2.6.1 Agents

Agents are autonomous entities who can decide next steps depending on their internal state and the state of the environment using set of roles [Ferber 1999][Wooldridge 2009]. The most important features of an agent in such systems are:

- Autonomy: When an agent starts its works, it takes its decisions independently of external control.
- Local views: as a bird or bee, an agent has no full global view of the system, or sometimes it is unfeasible to agent caches full knowledge.
- Decentralization: there is no central controlling agent.

2.6.2 Environment

Agent environments can be organized according to various properties [Ferber 1999][Vidal 2007][Potiron 2013] like:

- accessibility : The possibility of all agents to read all of environment sensors (variables).
- determinism: The action of an agent has a determined effect on the environment.
- dynamics: The effective number of agents that can alter the environment at any time.
- discreteness: The number of possible actions in the environment is known.
- periodicity: The effect of history of actions on the future.
- dimensionality: The effect of the location on the agent decisions.

The important characteristic of Multi-agent systems is the self-organization and other complex behaviors even when the individual design of all their agents is simple.

"The goal of multi-agent systems research is to find methods that allow us to build complex systems composed of autonomous agents who, while operating on local knowledge and possessing only limited abilities, are nonetheless capable of enacting the desired global behaviors" [Vidal 2007].

The interaction among agents often modeled by graphs where one or more agents reside inside nodes and they communicate over its edges. More detail about using this tool is given in 2.4.

2.6.3 Simulating a DCS

Simulation is a method used for experimenting designated policies and analyzing obtained results. The set of used objects with a predefined set of cause-effect relations form together what is called a model. For mathematical models, a computer is used to compute modeled system behavior. Results are analyzed, often statistically, to describe the impact of the developed policies.

Simulation is a popular technique used for improving or investigating process performance. It facilitates an evaluation of operating policies with considerable cost. In addition, it may be used to evaluate the performance of tools before trying them in production systems.

Simulation is used to analyze algorithms that are designed to work in distributed systems for several reasons, including working systems require adding suitable tools that may add additional overhead. Simulation provides number of users and resources that cannot be easily provided in the production systems. The debugging in the simulation will be less expensive.

Software tools (called simulators) are used to study many physical system types. Building such tools became a rich field of computer science in last centuries. To simulate distributed systems and multi-agent systems, researcher built many tools. A few significant and related tools are briefly described below.

- SimGrid: a scientific instrument developed to study the behavior of large-scale distributed systems such as Grids, Clouds, HPC, or P2P systems. It can be used to evaluate heuristics, prototype applications or even assess legacy MPI applications. The SimGrid project was initiated in 1999 to allow the study of scheduling algorithms for heterogeneous platforms [Casanova 2014].

- GridSim: a toolkit developed in the University of Melbourne that allows modeling and simulation of entities in parallel and distributed computing (PDC) systems like users, applications, resources, and schedulers (resource brokers). It facilitates creating various types of heterogeneous resources which can be managed by schedulers for designing computational algorithms [Buyya 2002].

- NetLogo: is a multi-agent programmable modeling environment. It is authored by Uri Wilensky and developed at the Center for Connected Learning and Computer-Based Modeling CCL. Netlogo is suitable tool for modeling complex systems developing over time [Tisue 2004][Wilensky 2014][Wilensky 2015].

- GraphStream: a graph handling Java library that focuses on the dynamics aspects of graphs. Its objectives is the modeling of dynamic interaction networks of various sizes [Pigné 2008][GraphStream].

  The library provides a way to represent graphs and work on it. GraphStream handles several graph types. It facilitates storing any kind of data attribute on the graph elements: numbers, strings, or any object.

Moreover, in addition, GraphStream creates a mechanism to track the graph evolution in time in terms of adding, updating, or removing elements like nodes, ages, and attributes.

GraphStream is built upon an event-based engine and its whole conception is object-oriented, each element of the graph (nodes and edges) but also each attribute that qualify these elements are objects. These features allow an easy prototyping and simulation of systems made of sets of interacting entities. In particular: (1) discrete-time simulations are easy to implement by associating an event to each time increment. (2) the object approach allows a decentralized point of view that prevails over centralized approaches for Complex Systems modeling since interactions between entities are mainly characterized by their locality. (3) finally, the library offers a lot of classical graph theoretic algorithms that may help for the analysis of interactions graphs.

2.7 Behavior analysis of a DCS using complex systems

The global behavior of the DCS emerges from interaction between its managing agents on multilevel of description. When DCS is self-organized, it exhibits characteristics of complex systems. Complex system theory is a methodology of science that studies how interaction between entities produces the collective behaviors of a system and how the system deals and forms relationships with its environment [Buchowski 1999].

2.7.1 Complex system

Systems are classified as simple, complicated, and complex. A simple system is described by one or several equations. It has input, process and output. The output is highly predictable if the input is known.

A complicated system is one composite from many simple systems with most of time linear relations. Difficulty of such system results from the large number of its parts, not the relation between them. In complicated system, reduction and decomposition of the system into subsystems can be made with presentations of the whole system i.e. they are modeled using top-down approach.

In complex system, it is impossible to reduce the system or to make decomposition without destroying it. They are modeled using bottom up approach.

Complex system does not own a generally acknowledged definition, however, most literature consider a system of many agents interact in special locality without central controller producing a global behavior that emerges from the interaction between the agents [Boccara 2004].

2.7.1.1 History

In complex system, we observe a group or macroscopic behavior emerging from individual actions and interactions. One of the oldest observations of complex

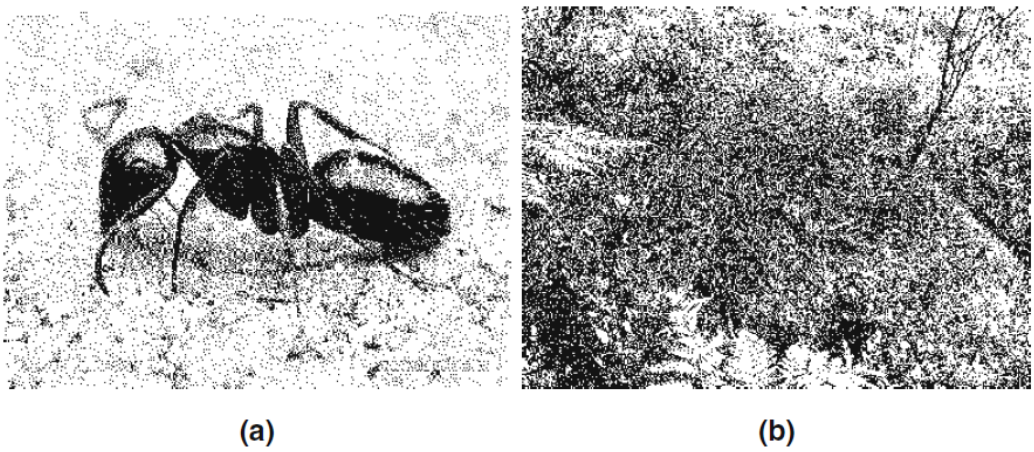


Figure 2.2: Ants: a single ant, b a colony. source: [Hoekstra 2010a]

systems was in biology, with the behavior of ant-colonies [Hoekstra 2010a].

Ants, termites, and some types of bees and wasps live in colonies. Behavior of such societies attracts the interest of the specialist of this domain. A member of these societies works according to limited knowledge. Every single individual in these colonies has a simple mission, at same time the colony is highly organized. Global tasks are completely accomplished without any central manager [Bonabeau 1999a].

Ants organize their life in a colony, see figure 2.2. Each ant performs a specific role such as foraging and nest maintenance. Ants organize themselves to perform tasks according to each task importance and the number of ants requires for carrying it out [Boccara 2004].

2.7.1.2 Principales

Complex systems have four main principles: decentralization, emergent behavior, self-organization, and the most important the adapted behavior.

Decentralization is a mechanism of operating a system where no parts of the system can be identified as a controller. System entities do tasks according to general rules that are understood by all without monitoring, rewarding, or punishment from any internal or external power.

Emergent behavior is an equilibrium state of the system that is hard to explain or analyze through aggregation of actions made by each individual. It is emerged from interactions between entities, without being described inside each entity.

Self-organization means many entities of same level of reasoning communicate, negotiate, and decide how to find a solution without an external intrusion for instance, the number of ants engaged in nest maintenance is relative to the damage size.

The main property of complex system is the adaptive behavior of the system like in ant systems: if the ant colony is partially destroyed, then the ants still in

live can adapt to regenerate the whole system behavior.

In general, complex systems identified if some of the following characteristics are observed [Boccara 2004][Érdi 2008]:

- It has large number of interactive entities called agents.
- It is composite from many complex and simple systems and the relations between them are non-linear.
- Often, a system has a (positive or negative) feedback from the historical decisions to keep the system in an equilibrium while, at same time, it makes the system state very hard to explain.
- Small change in the cause implies dramatic effects, with circular cause-effect relations.
- The global behavior of the system is emerging from the interaction of its parts.

Finally, use of the most important property is the adaptive behavior of complex system when the system, because of self-organization ability, can adapt from perturbations and can recover from failure without external control.

It is mainly this adaptive behavior property is of major interest concerning the DCS management model proposed in this thesis.

2.7.2 Computational aspects of complex systems

2.7.2.1 Cellular automata

Cellular Automata (often referred to by CA) is a discrete tempro-spacial model used in physics and computational sciences. Stanislav Ulam and John von Neumann were first originated the concept in the 19940s, see for instance [Chopard 2002] and [Hoekstra 2010b].

Cellular automaton uses integer indexes for both time and locations when referring to problem elements. Often the space of a cellular automaton is a regular grid of elements called cells. Each cell has a finite number of states (for example *on* and *off*). The grid can have finite number of dimensions. A cell is surrounded by a set of cells called the neighborhood. The future state of each of these elements is a function to its current state and the state of its neighbors.

CA has four main components need to be defined when building and implementing a model. They are:

1. state space defines the distribution of finite state automata (FSAs) on the regular d-dimensional lattice. For instance, in 2-dimensional lattice, a cellular automaton consists of a lattice of $m \times n$ squares where each square, often called a cell, is represented by one FSA.
2. cells values space is a finite set of discrete values that represent possible states of cells.

3. neighborhood defines characteristics of adjacent cells that affect and effected by a node state. For example, Neumann considered a neighborhood with radius equals one the four adjacent cells on top, right, left, bottom, and up while Moor considered the eight adjacent cells horizontal, vertical, and two diagonals.

4. transition rule is a function used to compute the new state of whole lattice. The new state of each cell is the value of the function and the input is the current state of cells located in its neighborhood.

5. time step is a discrete counter when increased all cells compute their new state synchronously.

What makes CA a very important is the feature of providing simple models for complex systems. Creating a collective behavior that emerges out from the sum of many simply interacting components became possible. Although, the basic on which local interactions is made simple, the global behavior shows new patterns that never be explained by aggregating properties of cells i.e. the total does not equal the sum of all the parts. Hence, cellular automata are an important approach used to model and simulate complex systems [Chopard 2002].

2.7.2.2 Ant colony optimization

The social behavior of insects and of other animals inspires researchers to develop a new approach to problem solving. Ant Colony Optimization ACO is the most successful optimization techniques that have been inspired from the foraging behavior of some ant species. It is noticed that these ants mark the path that should be taken by other members of the colony using a chemical material deposited on the ground called pheromone [Dorigo 2006].

A similar idea for solving optimization problems have been tried through Ant colony optimization. The mechanism described in ACO has been analyzed by premier authors: Jean-Louis Deneubourg, Eric Bonabeau, Guy Theraulaz and M. Dorigo in (for examples) [Bonabeau 1997][Bonabeau 1999b][Dorigo 2006] who implements the first ant-colony optimization algorithm (called ant system) in 1991 in his doctoral thesis. The algorithm finds its way for optimizing solution for many types of applications, for example routing and scheduling.

ACO mainly considers the environment as a graph where each time step, an ant moves from one node to another updating the *pheromone* value on the edge connected the old node with current one. For example, in the traveling salesman problem, a node represents a city and each connection between two cities is represented by an edge.

2.7.3 Dynamic graph

A graph is dynamic when its nodes, edges, or attributes are added, deleted, or updated over time. A dynamic graph is referred to with a time notation t as

$G(t) = (V(t), E(t))$ where, $V(t)$ and $E(t)$ are sets of vertices and edges at time t respectively. Each of these two set (or their attributes) may not be the same at time $t + \Delta t$, where $\Delta t > 0$.

A dynamic graph is used to model dynamic systems. Appearing and disappearing of system entities adds or deletes a node. Whenever a communication takes place between any two entities, an edge is created. It may be removed when involved entities are no longer interacting.

Structural features (measures and indexes) of such graphs may reveal important information about the self-organization process of solution development.

The goal of a dynamic graph algorithm is to develop efficient method of computing a feature after dynamic changes, and not recomputing it from scratch each time [Pigné 2008][Demetrescu 2010].

Since there is non-easy solution to compute features of dynamic graph, some tools are developed for this purpose. Pigné et. al. [Pigné 2008] built the Graphstream tool to model dynamic complex systems. Authors showed that a group of clauses in form of (date, events) could be used to define a dynamic graph. One or more graph entities are modified with every single or set of clauses.

In DCS, an overlay network, which presented below, is modeled by a dynamic graph.

2.7.4 Overlay network

An overlay network is a logical representation that is created from events (information exchange or actions) frequently takes place between system nodes. The notation is widely used in Peer-To-Peer P2P network, see for example [Kshemkalyani 2008]. Overlay network is created over a one used to diffuse the information called underlay network. Underlay network is a static network while the overlay network is dynamic (at least the changing rate of the former is quite smaller than the latter).

Many overlay networks may be considered through classifying a set of gathered information into different categories. For example, files may be classified by location, ownership, type, etc. When no overlay network is defined, nodes are considered isolated or not interacted. The underlay network should be connected to have information delivered to maximum possible number of nodes. Overlay network may be not connected at given time but it should be connected during given interval of time. It is may be called *interactively connected*. Three nodes A, B, and C are interactively connected when:

At time t an interaction takes place between A and B, and *if and only if*,
at time $t + \Delta t$ another interaction takes places between B and C, where $\Delta t > 0$,
such that an action made between B and C is affected by one made between A and B.

It is important to describe the structure of overlay network that supports the stability of the DCS. Overlay network structure characterizes a dynamic state in

which the system is self-organized.

The structure of the overlay network depends on many parameters like structure of the underlay network, discovery method, information aging, and the capacity of local cache or the buffer used to handle delivered information.

The underlay network could be physical and not easy to reconstruct. Each discovery method has its own basics and complexity and produces different overlay network structure. Information aging follows the changing rate of state of elements that are described by the information. The validation of information is measured by its recentness. When the acceptable age of information is very high, a complete graph may be constructed after several iterations of resource discovery i.e. each node becomes know about the state of all other nodes. The capacity of local cache determines the maximum outdegree of the graph representing the overlay network. The small limit of the capacity of local cache makes no need to use large age of information.

In this thesis, we find that (which is one of its major contributions) any change made on the content of the local cache of a node changes its degree in the overlay network. If the change is made systematically according to some preference, the structure of the overlay network may take a form of well-known structures like a power-low distribution or scale free form. This self-organized structure improves the global performance of the system. It also produced by a self-adaption of nodes in the resulting structure without explicit encoding in the behavior of each node.

Chapter 3

PROBLEM SOLVING METHODOLOGY

Contents

3.1 Architectural model of DCS 34
3.1.1 Node description 34
3.1.2 Agent description 35
3.1.3 Network types 37
3.1.4 Workload description 37
3.2 Resources discovery 40
3.2.1 Load balancing 40
3.2.2 Improving information management 41
3.2.3 Reinforcement 41
3.2.4 Information preference 41
3.3 Heterogeneous system 42
3.3.1 Nodes with different architectures 42
3.3.2 Nodes of multi-core 43
3.3.3 Communication bandwidth 43
3.4 System state indicators 45
3.5 Interaction capturing model 46

One of the main characteristics of complex systems is that it can be described through multiple levels system. Relations exist between entities at each level as well as between levels. A DCS is modeled as multiple-level system composite of nodes and network. Nodes are computing units at basic level. Network is the facility that delivered communication messages between nodes. A node may host several agents. Agents are managing programs to which tasks are assigned. They evaluate the state of the system through predefined measures. When agents communicate through some type of connection, a network is considered. An agent controls its behavior autonomously depending on its own knowledge. The global behavior of the system emerges from the interaction between agents and the feedback evaluated from the historical events, which is another characteristic of complex systems. Workload is the input for the system. It should be processed and results retained to the submitter with lowest operational cost.

3.1 Architectural model of DCS

3.1.1 Node description

Let a distributed computing system DCS contain a set of vertices (called nodes)V:

$V = \{v_i : v_i \ is \ a \ computing \ node, \ i = 1, 2, ..., N\}$, where N is the initial number of nodes.

Nodes differ in their architecture according to their creation purpose. Architecture means the ability of a node to execute a specific type of jobs. Some nodes are considered generic since they have architecture able to serve all job types.

Let $A = \{A_k : A_k, \ k = 0, 1, 2, ...K\}$, be the set of architectures,

where K is a predefined number of architectures that are supported by the system. A_0 is the generic type,

A node v_i has C_i CPU-cores. A core is the processing part of a CPU chip without cache. It composites of the control unit and the arithmetic logic unit (ALU).

A node can communicate with another node when the address of the latter is known and the communication is authorized. Both the knowledge and the authorization are modeled by a link between nodes. As a result a network is created and it is often modeled by a graph $G = (V, E)$, where:

$E = \{(x, y) : (x, y)$ is a link between node x and node y, $x, y \in V\}$.

A node $v \in V$ connected to a set of nodes $E_v \subset E, E \neq \phi$, which is called its neighborhood.

Let a workload $W = \{j : j$ is an atomic job$\}$ be a set of jobs that are submitted to DCS during a duration defined by T cycles. T is the duration of time during which the system is up and receive jobs. A cycle is a representation for the unit of time.

The job inter-arrival time is a random variable that often follows a statistical distribution based on the analysis of traces from production systems.

Nodes use same distribution but they vary in the scaling parameters. More detail about workload models are explained in 3.1.4.

The load state of a node, L_v equals the queuing time for a new submitted job

plus the remaining service time of currently running jobs. L_v is measured in cycles.

Load varies from one node to another according to the variation between them in the job arrival rate, the number of cores, and the architectures. Some nodes are highly loaded and others are lightly loaded. Hence, collaboration is the main objective for building DCSs. Lightly loaded nodes participate doing some work for highly loaded nodes by means of job migration between them to keep the system in an equilibrium state.

Load information is modeled by descriptors. A descriptor, denoted by d_i, of a node v_i is a formatted set of values. It contains both physical and state information. The features of descriptor that are used in our model are explained below.
Let $d_i = \{Id,\ C,\ A,\ activeness,\ QueueLength,\ WaitingJobs,\ timestamp\}$, where:

- Id: the identification of the node to which the descriptor belongs.
- C: number of installed cores in the node.
- A: the hardware or software architecture that is supported by the node.
- $Activeness$: the difference between number of imported and exported jobs by means of load balancing process. It is the activity history of a node.
- $QueueLength$: queuing time for a new arriving job plus remaining service time.
- $WaitingJobs$: number of jobs waiting in the queue.
- $TimeStamp$ is the time when this information is registered. It is an indicator used to validate received descriptor.

Node v_i caches collected descriptors in a limited size list denoted by D_i. D_i is maintained frequently to keep its content consistent.

As seen, a node does many tasks at the same time. It is preferable to have several agents that do tasks simultaneously for a given node. Agent modeling is described in details in 3.1.2.

Agent interaction takes place on node level and on network level. As a result, new networks emerge. Information exchange creates and develops an overlay network. Job movement between nodes in term of load balancing creates another network called migration network. The models of these networks are given in 3.1.3.

3.1.2 Agent description

A node hosts a set of agents each one is assigned a task from the set: $Tasks = \{$ *scheduling, communication, discovery, balancing, forwarding* $\}$. Below, a definition of each agent and its task is given.

- Scheduler, denoted by S, receives arrived jobs and schedules them for executing. An arrived job has to enter in a queue, denoted by Q, waiting its turn of execution, or it has to wait in another queue, denoted by $Ex - Q$, for migration to another node when some of the requested resources cannot be provided locally.

- Promulgator, denoted by P, communicates and exchanges information in form of descriptors with neighbor nodes in specific network. Descriptors are cached in a limited size list D. P may uses several diffusion methods, see section 3.2.

- Mobile agent (optionally), denoted by M, is a roaming object. M moves from node to node like a bee. P usually welcomes M and they exchanges up-to-date descriptors. M transfers its self to one of current P's neighbors that is chosen at random.

- Balancer, denoted by B, applies a selected load balancing strategy denoted by s, see section 3.2.1. B uses local information cached in D to trigger load balancing process. It determines the source and destination for the migration. An initiator B contacts the corresponding B at another node to move jobs between them.

- Forwarder, denoted by F, forwards jobs from $Ex - Q$ to a compatible node chosen from D.

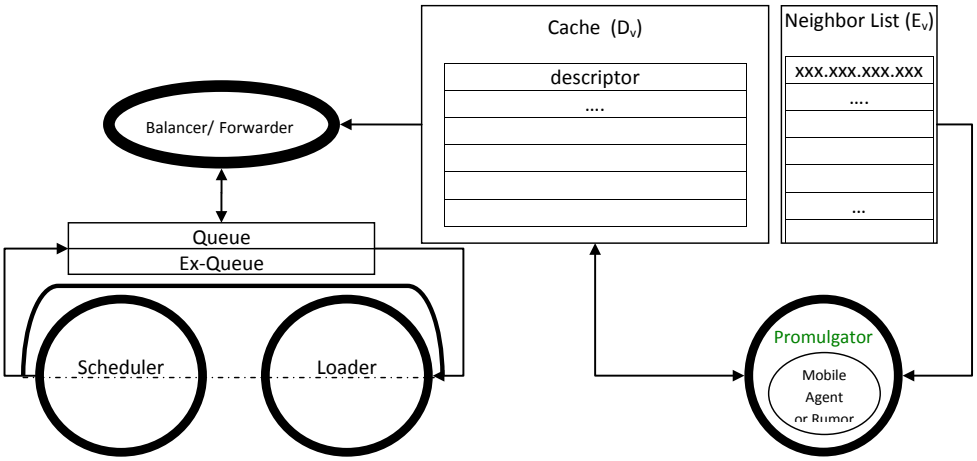


Figure 3.1: Relationship between local agents

Agents cooperate to achieve a global task as follow: Migration service agent uses raw information provided by discovery service agent. The former updates the status and parameter of the latter and spreads information by means of migration

dialog. Discovery service agent filters information using the parameters provided by migration service agents. Hence, the structure of overlay network and migration network becomes dynamic and complex. The performance of the global system is enhanced when the mean response time MRT is minimized. MRT depends on workload and overlay network characteristics. We cannot change the workload characteristics, but we can for the overlay network using cooperation mechanism, as explained in 3.2. Figure 3.1 shows the relationship between some local agents working in a node as a tractor with trolley. Starting from the front of the tractor, the scheduler receives submitted jobs from users, puts them in appropriate position in waiting queue. Loader is a part of the scheduler that allocates available resources for next job in the queue. Promulgator prepares the descriptor of its node, selects a node from the list of direct neighbors, exchanges, and caches information in D. Balancer contacts a node (or several) chosen from the cached information for job migration when load variation is detected.

3.1.3 Network types

The network is the basis in any distributed system. A network has many meanings. It could be physical or logical (depending on link definition), static or dynamic (depending on link stability).

According to the different link definitions, we got different network types and structures. The types of graphs (networks) relevant to our model are:

- Base graph G_B: used to model the underlay network (physical direct connections, or logical as routes) of a distributed computing system. G_B reflects predefined connections between system nodes. G_B is an input for the model and remains unchanged during system life. It can be directed or undirected. Two nodes linked by an edge are called neighbors.

- Overlay graph G_O: Information cached at each node forms this graph. An arc represents an entry in the cache. Arcs point from current node toward nodes present in its cache. G_O structure is dynamic and affected by discovery method, local cache size, and the validation rule of information.

- Migration graph G_m: a directed weighted graph that evolves through job migration flow. An arc exists from node a to node b if at least a migration from a to b exists during the system live. A weight attribute is used to assess the repetition of migration event for a given arc. This graph is affected by the structure of overlay graph G_O (or G_B if no overlay is used), load variation between neighbor nodes and load balancing strategy. Figure 3.2 shows the relation among the three networks.

3.1.4 Workload description

Models need many characteristics for performance evaluation: job arrival rate, run time , number of required processors (cores), etc. Each of these characteristics

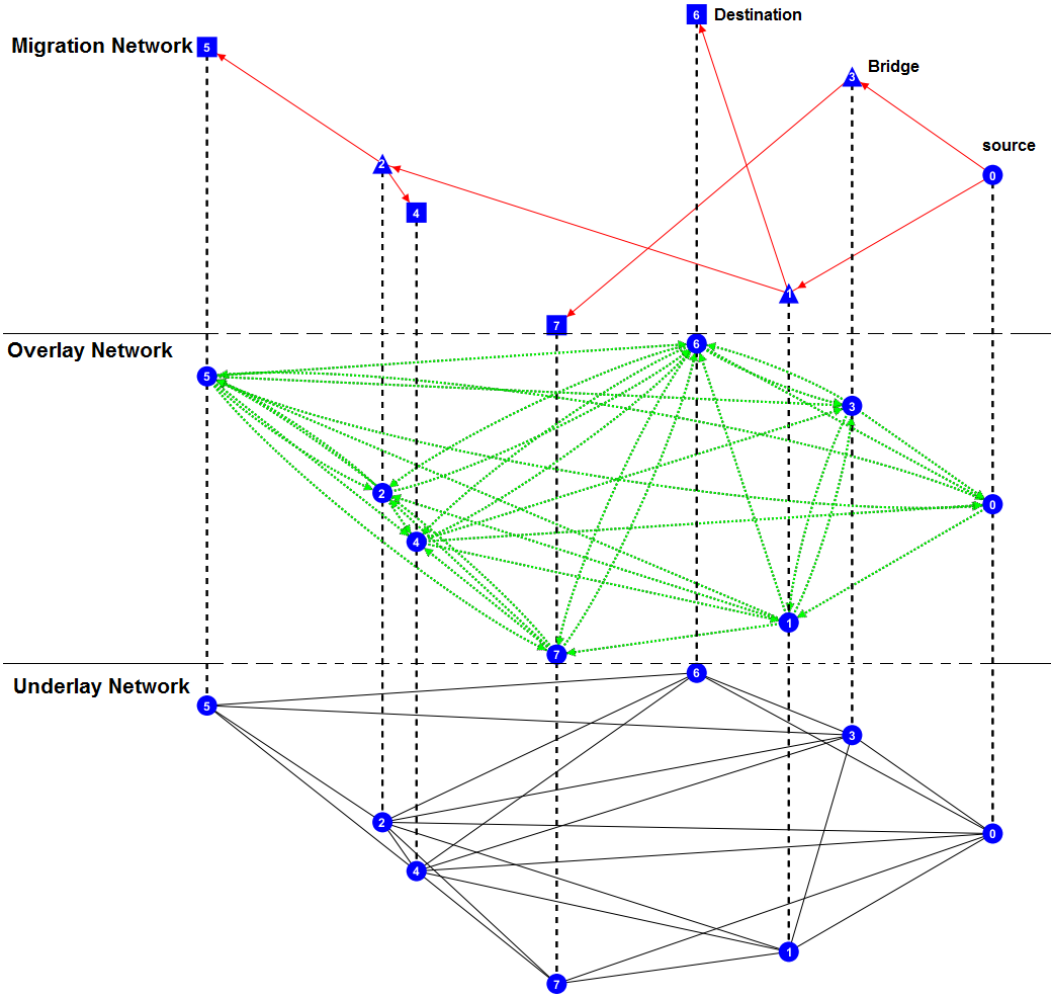


Figure 3.2: Multi-layered Network

(and others) is usually sampled from a specific distribution. Some models that are proposed in papers are briefly reviewed below.

Lublin and Feitelson odel[Lublin 2003] built a model that generates workload including three components:

- Jobs arrive following a two-Gamma distribution:
- Run times follow a hyper-Gamma distribution using six parameters.
- number of processors (called job size in the paper) is modeled based on a two-stage uniform distribution with four parameters. Two parameters specify the minimal and maximal required size, and the other two parameters determine the percent of serial and power-of-two jobs. The different between serial and power-of-two job is that the latter request number of processor p can be written as power-of-two $p = 2^k$.

Li et. al. model [Li 2005] proposed five distributions for some characteristics of workload.

- Interarrival time: gamma or two-phase hyper-exponential for peak load periods and Weibull for the other periods.
- Canceled jobs distribution: log-normal is suggested as best suitable.
- number of processors (called job size): the most suitable distribution is loguniform.
- Actual runtime: Weibull or lognormal is the best-fitted distribution.
- Actual runtime conditioned on requested time ranges (R): The best fit is obtained from: two-stage loguniform for small R, Weibull for medium R, and lognormal for large R.

Li et. al. [Li 2007] presented an initial analysis and modeling of Grid job arrival patterns. Author modeled job arriving process through a set of m-state Markov Modulated Poisson Processes (MMPP). A comparison is also made with Poisson processes and hyper-exponential renewal processes as alternative models.

Dror G. Feitelson has a book on workload modeling [Feitelson 2015]. The book focuses on the process of using workload traces and deriving synthetic models.

In the following sections, we give the chosen models that are used to assign job characteristics.

3.1.4.1 Job arrival rate

In a system composed of N nodes, jobs arrive at any node independently of other nodes.

The number of jobs that are expected to arrive to such system in any cycle is given by **Poisson binomial distribution** which is a discrete probability distribution defined as the sum of **independent Bernoulli trails** that are not necessarily identically distributed[Daskalakis 2015]. Local arrival rate λ_i is he mean number of arrivals in one cycle for node i. λ_i is generated uniformly in the interval $[0, 2\lambda]$, where λ is the global mean arrival rate. According to queuing theory, it is necessary to have a mean arrival rate strictly less than the mean service time or the system will not be stable, i.e. $\lambda < \mu$.

3.1.4.2 Job length (run time)

Two models are used to generate job run time: constant and variable. The constant model assigns μ run time for any arrived job. Variable model samples run time from the exponential distribution with mean μ.

3.1.4.3 Job architecture

The simplest way is to create a hash table of all distinct possibilities of architecture. Then, sample index of given architecture following normal uniform distribution.

3.1.4.4 Job Size

By inspecting many file systems, it is found that file size follows exponential distribution. Since a job is single file or set of files, its size also follows an exponential distribution. Likes job run time, constant and variant job size are used.

3.2 Resources discovery

An agent needs to know about the state of its neighbors to cooperate with them. Many methods exist for information exchange. Three methods are tested here.

- Local scheme: information agent, I, periodically queries all direct neighbor nodes in the initial network. The communication complexity is $O(|E|)$, where $|E|$ is the number of edges in the initial network. This method is used for static and "small" networks that have small average degree.
- Rumor spreading: information agent, I, exchanges the content of local cache with a neighbor node that is chosen at random. The communication complexity of this method is $O(|V|)$, where $|V|$ is the number of nodes in the networks.
- Mobile agent: information agent, I, welcomes a visiting mobile agent. They exchange cache content. A mobile agent transfers itself to a neighbor node that is chosen at random. The communication complexity of this method is $O(M)$, where M is the number of mobile agents. m is either static or dynamically controlled by the nodes.

Information in the local cache represents the partial "view" of a node to its system global state. In the first method, each node has a complete knowledge of its neighbors, but that is all. In the two other methods, a node has a partial image of the complete network. These methods are called global scheme. Agent I maintains its view periodically.

3.2.1 Load balancing

Four different load balancing strategies that are presented in 2.5.4 are considered in our model. In Sender Initiated Diffusion (SID) strategy, the source node selects the destination node at random. In Receiver Initiated Diffusion (RID) strategy, the destination node selects the source node at random. In Sandpile based strategy, the source node selects two destination nodes at random. Finally, we proposed Help Local Maximum (HLM) strategy in which the destination node selects the source node that is most loaded node.

Several options are selected for choosing a job for migration: (1) Latest job in the waiting queue of the source node is selected to minimize the global MRT. (2) Best fitted job (architecture or number of cores), in heterogeneous systems. (3) Smallest size job to minimize bandwidth usage. (4) shortest execution time to minimize waiting time of short jobs.

The selected job is migrated only when the migration will reduce its response time, and the load inequality between the source and the destination nodes does not change after the migration. Balancer agent B monitors the load status of its node. It initiates migration process when the initiation policy is applicable.

3.2.2 Improving information management

Load balancing depends on information provided by resource discovery. Load balancing may participate in spreading information by means of migration dialog. Resource discovery may provide the information that is more preferable by load balancing. Hence, some mechanism of cooperation between the I (information manager) and B (balancer agent) is added. This cooperation changes the overlay structure and enhances obtained results.

3.2.3 Reinforcement

B initiates load balancing with a node that is chosen from local cache according to selection policy. Actually, B uses uncertain information, since local cache may have old items, i.e. their age $\simeq TTL$ and $TTL > 1$. The initiator starts communication with selected partner by asking it for its current state. It adds received information to its cache with $age = 0$. Then, I spreads this "fresh" information to neighbors. As a result, candidate partners become known in area out of TTL distance and may be selected by more nodes.

In fact, these nodes characterized by extreme arrival rate (nodes with very high arrival rate are demanded by receiver initiated scheme and nodes with very low arrival rate are demanded by sender initiated scheme). Hence, for each selection, a new promulgation region is created. Highly demanded nodes become hub nodes in the overlay network. We called this mechanism *reinforcement*.

3.2.4 Information preference

When the number of collected items is larger than cache size K_D, extra entries need to be dropped. Dropping is made after sorting entries by some preferred order. Then, top K_D items are kept and dropping extra entries. By default, cache items are ordered by their recentness, i.e. their age.

Load balancing prefers nodes that did success in most previous migration process. Load balancing sets a variable called *activeness* which is added to node descriptor fields. Activeness value equals to the difference between number of sent jobs and number of received jobs that are migrated by means of load balancing. The activeness measures the participating in the load balancing process.

I sorts cache items by activeness in ascending or descending order according to weather a sender or receiver initiated diffusion load balancing is used. SID uses ascending order and RID uses descending order. I drops any oversize (less important) items.

3.3 Heterogeneous system

So far, we dealt with a homogeneous system that is composed of identical nodes. In real life systems, this case is rare. A node in our model is a representation of individual servers, clusters, small grid, or in general computing node with any Internet connection. Three relevant features having most effect on cooperation between computing nodes are considered: architecture, bandwidth, and number of cores. Below, description of using each feature in our model is given.

3.3.1 Nodes with different architectures

Community refers to a property that a group of nodes shares. It may refer to location, organization unit, ability of doing some types of jobs, etc. Nodes are distributed in the Internet according to some organizational issues as in industrial centers and scientific institutes. A node may have special software packages; it may execute only special types of applications. A node belongs to generic community when it is able to serve all types of jobs. As a result, a node in such network has either generic or specific community label. Nodes may organize themselves into one community or different communities in the overlay network according to weather their cache contain a mixed or one type architecture.

3.3.1.1 One community overlay structure

When local cache contains information about nodes of any architecture, an overlay network is created with all nodes belong to one community. We call this an overlay of "simple" structure. When imbalance is detected, only compatible nodes having information in local cache are eligible for sending or receiving jobs to or from current node.

The scheduler places only jobs that are executable in its node in the waiting queue Q. If arrived job requires resource of different community, a *forwarder* agent denoted by F redirects that job to a compatible node. When more than one compatible node is found, a minimum loaded node is chosen. However, finding a compatible node in mixed-content cache is not always possible. Hence, the redirection operation may be differed for next cycles when F does not find any compatible node in local cache.

3.3.1.2 Multi-community overlay structure

To solve the problem of finding a compatible node for job migration, we propose a method to construct an overlay network having a "Community structure"[1]. We call this overlay structure "complex'.

The proposed method works as follows: When information is diffused, information agent, I, caches information belongs to nodes of the same or a generic community only. That is, nodes of same community become highly intra-connected forming a homogeneous cluster. Nodes of generic type act as joint between homogeneous communities. They may (potentially not) provide services (executing jobs) but surly they participate in load balancing.

However, only generic nodes have to be aware of job-to-node compatibility when migrating jobs with non-generic nodes. Figure 3.3 shows a schematic structure of such complex overlay network or a multi-community overlay network. Most migrations are done inside each community. A job submitted to non-compatible node will be forwarded to a generic node. From there, it may be executed or migrated to a compatible node.

3.3.2 Nodes of multi-core

Most nowadays systems are supplied with many cores (CPUs). The concept of virtual CPU, virtual core, or even a virtual processor is widely used. It means a part of physical processing power is assigned to a virtual machine (VM). In general, either a node has multi-physical cores or it implements a symmetric multi-processing (SMP) multi-threaded computing model to provide multi-core system.

However, no resource sharing like cpu is considered in this study. At any time, a node can execute only one job.

3.3.3 Communication bandwidth

Internet is the common means of communication between any two nodes in the world. Any two nodes that are connected to the Internet can exchange messages (information or jobs) unless restrictions are specified. Two issues are related to communication process: Latency and bandwidth.

Latency is the time required for the first bit of the sent message to arrive at its destination. Latency depends on transmission media (for example fiber-optics vs. wireless) and its current usage (depending on number of messages being transmuted by other process at current time). However, the time taken by preparing and receiving processes may be also added for the latency[Coulouris 2011]. In our model, a constant latency is considered, where a message takes one unit of time to move between any two nodes.

Bandwidth means the maximum number of bits can be transmitted over given link. Nodes (or processes) that use same network connection have to share the

[1] According to Newman[Newman 2003]: "groups of nodes that have a high density of edges within them, with a lower density of edges between groups " exhibit a community structure

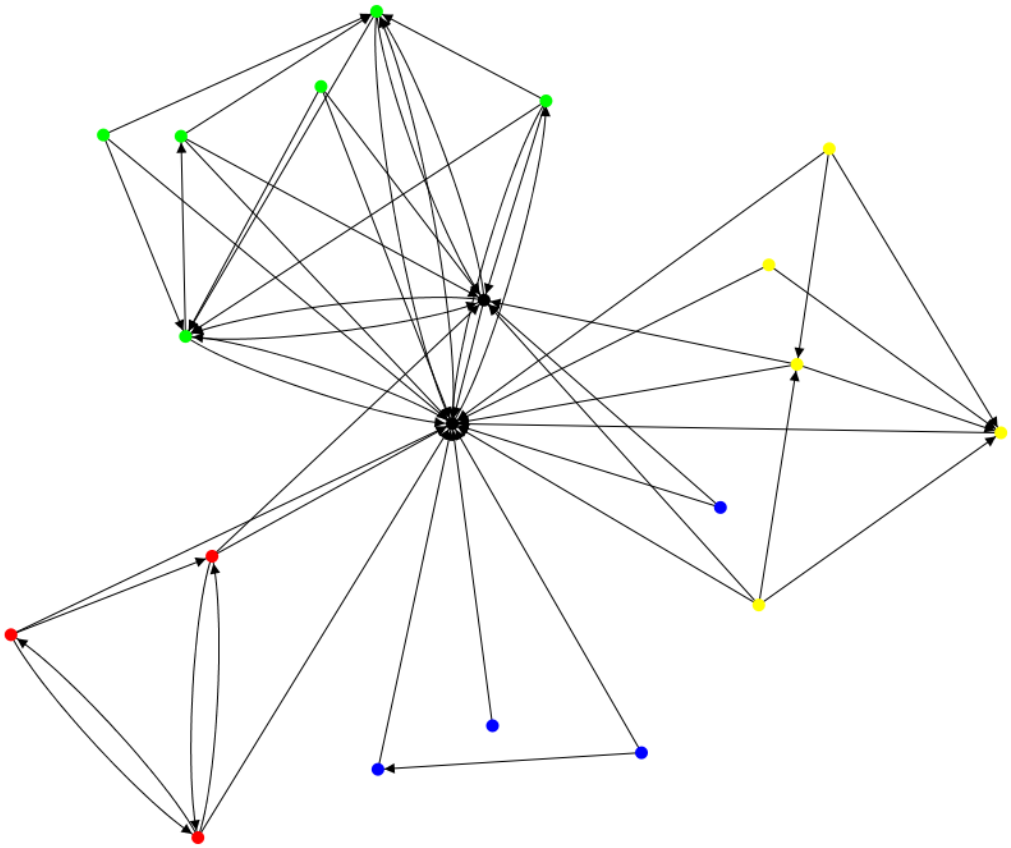


Figure 3.3: Scheme of a multi-community (architecture) overlay network

available bandwidth. The connection bandwidth is allocated by a contract with the Internet Service Provider ISP. Hence, nodes belonging to different domains vary in their bandwidth. However, allocated bandwidth could not exceed the designating bandwidth of communication equipment.

A node is connected to the network (Internet) by a link with a bandwidth that is large enough to send as many as required number of messages per time unit. However, each message takes different duration for moving from source to destination. Two modes of bandwidth usage are considered: identical and different. In identical mode, all nodes have same connection bandwidth. Any message takes only one unit of time.

In different bandwidth mode, each node is connected to the network with its own bandwidth.

Typically, information messages are small. Hence, it takes at most one single unit of time to move between nodes.

Job migration takes duration of time according to its size and the bandwidth of both source and destination. Migration time is calculated by the formula:

$time_j = \lceil \frac{size_j}{min(B_{src}, B_{dst})} \rceil$

where $time_j$ is the estimated transmitting time, $size_j$ is job size in data unit, B_{src} and B_{dst} are the Internet connection bandwidth of source and destination nodes respectively.

3.4 System state indicators

Statistical measures on the workload of nodes are used to assess system stability. Some of used measures are defined below starting by presenting some useful notations.

- Let p_j be the length (its processing time) of job j.
- Let r_j be the submitting time of job j.
- Let c_j be the completion time of job j.
- Let L_v be the load of node v. The load is measured by the remaining time of jobs being executed, plus the execution times of waiting jobs.
- Mean load: denoted by $\overline{L}$ represents the arithmetic average of all nodes' load. $\overline{L} = \frac{\sum_{v \in V} L_v}{n}$.
- Standard deviation of queue length: denoted by σ_Q, a measure used to assess the degree of balancing of the global system.
  $\sigma_Q = \sqrt{\frac{\sum_{v \in V} (L_v - \overline{L})^2}{n-1}}$.
  The load L_v of a node is used to decide of a migration
- Response time: denoted by R_j, duration of time starts from submission time of that job, r_j, and ends at completion time, c_j, of its execution (or when first response is received from the job in interactive environments).

  $R_j = c_j - r_j$.
- Mean response time MRT: denoted by $MRT(t)$, is the arithmetic average of response times of all jobs that are terminated at time t or during the interval $t = t_2 - t_1$.

  $MRT(t) = \frac{\sum_{j \in J} R_j}{J(t)}$, where $J(t)$ is the number of all jobs that complete their execution in time (interval) t .

The performance of the global system will be measured by the notation of MRT during a given time. The system reaches a steady state if and only if MRT keeps the same value for long time.

3.5 Interaction capturing model

Exchanging messages between actors is considered as an interaction. Frequent interaction between two entities is modeled as a relation between them. A relation is often modeled by a graph where nodes represent the actors and links represent the relation. In a DCS the actors are the computing nodes (or their agents) and the message may contain load information or a complete job if it is migrated. Our goal is to build graphs (static or dynamic) to capture these interactions, and thus explain the system performance and behavior. Three elements should be explained when building interaction model:

1. *Event*: the type of an interaction between actors. For clarity, we focus on one type and omit non-related interactions. An event could be general as message passing or specific like job migration.

2. *indicator*: a variable used to handle or assess the occurrence rate of given event between actors. Two indicators are proposed:

   - Spatial: The simplest indicator is a counter. The counter is incremented at each occurrence of the event (each job migration for example) through a given arc. Time is not taken into account.
   - temporal-spatial: like the pheromone in ant systems. The indicator value is incremented for each migration and is decremented at fixed rate with time. The value of such an indicator is a function of two variables: event occurrence and time. Pheromone is updated in ant system by the following formula.
     $p_{ij}(t+1) = \rho.p_{ij}(t) + \Delta p_{ij}(t, t+1)$, where
     i, j are the ends of given arc.
     $p_{ij}(t)$ is the concentration at time t.
     ρ is the evaporation rate.
     $\Delta p_{ij}(t, t+1)$ is the quantity per unit of length of trail substance (pheromone in real ants).

     Same formula is used in a DCS model.

3. *The threshold*: A filter used to keep robust relations only. Two type of threshold are proposed.

   - local threshold: for each node, only one arc that has maximum indicator value is kept. All other arcs are removed.
   - global threshold: All arcs that have their indicator value less than T are removed, where T is a global threshold selected from the distribution vector of the indicator.

In detail, the model is implemented as follow:

A *migration* graph G_m represents job flows from overloaded to under-loaded nodes in the network. It is initialized as $V(G_m) = V(G_B)$ and $A(G_m) = \phi$. The *event* here is the job migration.

While the system is running, for each migration from u to v, if arc $(u, v) \notin A$, then it is added $A \leftarrow A \cup (v, u)$.

(v, u) has an attribute called either "counter" or " pheromone". This attribute is the *indicator* of migration event. $w(v, u)$ is updated for each migration by one of proposed methods.

A node $v \in G_m(V)$ has an attribute called activeness a. $a \leftarrow (J_{in} - J_{out})$, where J_{in} is the number of jobs brought to v and J_{out} is number of jobs taken from v.

When the system is stopped, or when needed, a snapshot of the migration graph is taken for inspection. The following actions are made on given snapshot G_m:

1. each node $v \in V(G_m)$, let $v_{out} > 0$ and v_{in} be node's outdegree and indegree respectively, then v is called :

   ⊡ *source* and shaped by a rectangle: if $v_{out} > 0$ and $v_{in} = 0$,

   △ *bridge* and shaped by a triangle: if $v_{out} > 0$ and $v_{in} > 0$,

   ◯ *sink* and shaped by a circle: if $v_{out} = 0$ and $v_{in} > 0$.

2. A selected *threshold* is applied by deleting all arcs having their indicator is less than T. T is empirically chosen to break migration graph into a significant number of components.

3. Common indices and measures of resulting components are calculated , for example: number of nodes, number of arcs, average node degree, diameter, number of loops, strongly connected components, weakly connected components etc.

Similarly, the overlay graph is built but it is a dynamic graph, as the content of the cache is constantly updated. In next two chapters, we simulate our model and discuss results obtained for different configurations and schemes.

Chapter 4

THE SIMULATOR

Contents

4.1 Introduction 49
4.2 Network 49
4.3 Node implementation 50
4.4 Agent implementation 51
4.4.1 Network agents 52
4.4.2 Node agents 57
4.5 Workload implementation 62
4.6 Summary 63

4.1 Introduction

A discrete-event simulator has been built using Java language. The cycle is its time unit. GraphStream is chosen as base for our simulator since it supports dynamic graph where nodes, edges, and attributes are added and removed easily. It is available as Java libraries which make it flexible to implement required abstraction level.

The simulator is used for testing resource discovery methods and load balancing strategies. It is built hierarchically with two levels. The top level represents the networks. The base level represents the nodes.

4.2 Network

The simulator handles a network as an object instantiated from the "Graph" class. The network itself is viewed as a three levels network: Base, overlay, and migration networks. For each network there is a corresponding graph:

Base graph represents design scheme of a given DCS. For each known DCS there is a graph that models the corresponding network. For the simulator, Base graph is a file gathering all information on the instance network. The Base graph is static during the time of the simulation. It is obtained either from the Internet (for example [CADIA]) or it is generated according to theoretical models, see 2.4.

The overlay graph emerges from base graph by the interaction that takes place between nodes in term of information exchange and caching. The overlay graph

is directed and dynamic since the local cache is dynamic in term of number and source of its items. We can have a snapshot for the overlay network anytime during the run of the simulation. The resource discovery method and agent interactions affect the structure of overlay graph during the simulation.

The migration graph is a model of interactions in term of job movements. It is directed and an arc in this graph corresponds to a migration of jobs from one (source) node to another (destination) node. Its arcs are weighted by a value that indicates the density of migrations made over each arc. The structure of migration graph depends on the overlay graph, since the migration process depends on information available at nodes.

Base, overlay, migration graphs, use the same set of nodes but different edge (arc) sets for each graph model.

4.3 Node implementation

Node's information is implemented by a class called the "*descriptor*". A descriptor defines attributes that are needed for resource discovery. Below, we list these attributes. Note, only the first three attributes are physical description of a node, the others are status variable.

- *id*: the identification of the node.
- *arch*: the hardware or software architecture that is supported by the node.
- *bandwidth*: The speed of the link used to connect a node to its ISP (Internet Service Provider).
- $Waitingjobs$: number of jobs waiting in the queue.
- *queuelength*: queuing time for the waiting jobs plus remaining service time of the job being executed.
- *activeness*: the difference between number of imported and migrated jobs. It records activity history of a node.
- *timeStamp*: the time when this information is created. An indicator used to validate received descriptor.

A node itself is implemented by a class called *Host* which wraps up both resource and agents. The Host class is an extension to descriptor class. Many other structures have been implemented in a node for operational, monitoring and managing purposes. It also provides a suitable environment for agents to do their tasks. Below, some notations used for these data structures are defined.

- Q is the waiting queue to be used by the scheduler for handling jobs that are scheduled for the processing.

- Q' is the forwarding queue where non-compatible jobs are handled while the forwarder agent searches for them a compatible node.
- D is the descriptor table where up-to-date information is cached.
- D' is the input buffer used to handle descriptors for other nodes delivered to the node by resource discovery method.
- A is the buffer where arriving mobile agents are handled.
- A' is the buffer where leaving mobile agents are handled.
- J' is the buffer where arriving jobs are handled (new submitted or migrated by load balancing process).
- grp is a vector of nodes that contains the identification of one source and one or two destination(s) nodes, which are elected depending on available information for load balancing, see 14. Note: the first element always refer to the source node while grp is equal $null$ when the node does not require to initiate load balancing process.

4.4 Agent implementation

The simulator is also built as an agent-based system. Hence, in addition to managing agent that are defined in 3.1.2, other agents are also implemented to have a working simulation environment. For example, see below, each between-nodes operation is managed by a dedicated agent while such operation is the network responsibility in a real implementation. Therefore, distinguish should be kept in mind between model agents and simulator agents.

An agent is an active object. Its state is represented by object attributes while its behavior is defined by a set of methods. A special method run() is activated autonomously when its state is changed. The simulator has the ability to install any number of agents that is derived from abstract agent class "Agent".

An agent is implemented as a "Runnable" class in java. Hence, it could be started once as independent thread. But this depends on the architectural properties of the used computer system. It is also possible to call agents asynchronously by using the set structure by arbitrary iterating them one by one for execution.

Two types of agent are defined: network agents and node agents. Network agents work at network level and perform actions like information spreading and jobs delivery between nodes. Nodes interface with external world through buffers. Local agents receive information and jobs from input buffers, process them and put result (if any) in output buffers. Figure 4.1 shows the communication scheme between local and global agents.

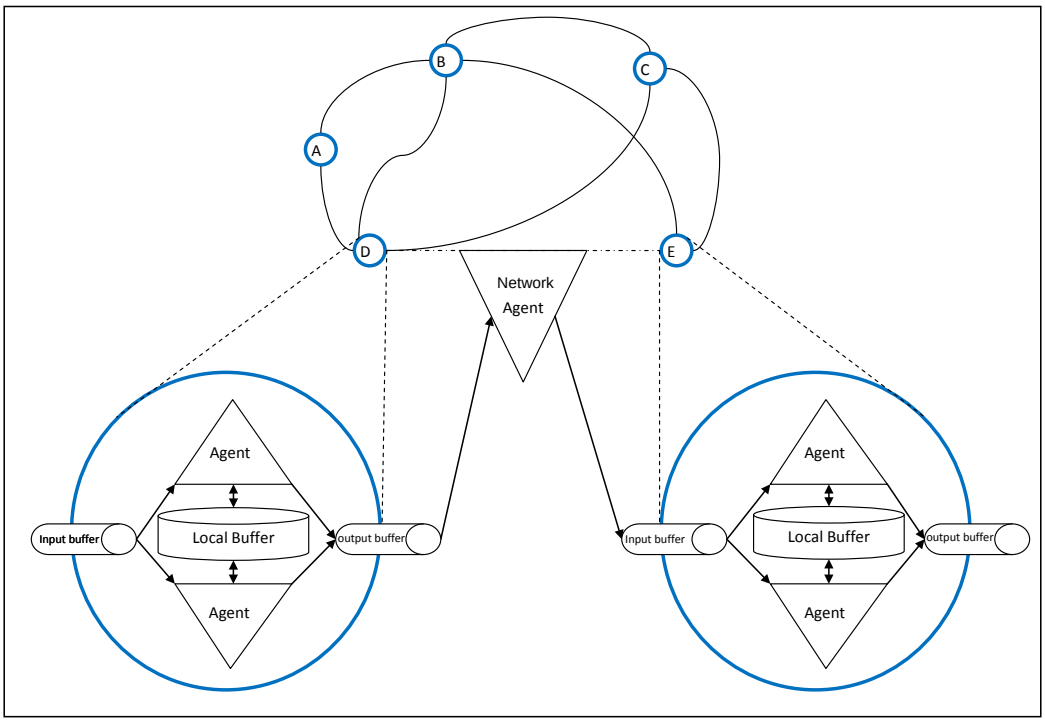


Figure 4.1: Network and node agents communication scheme

4.4.1 Network agents

The network agent is a simulator related agent. It delivers either information (called diffuser) or jobs (called migrator) e.g. two network agents are built for simulating the communication between node agents in different nodes.

Diffuser is implemented using three, depending on the chosen resource discovery schemes:

4.4.1.1 Local Diffuser

Local performs the local scheme of the resource discovery. It exchanges load status between any two nodes that are connected by an edge in the base graph i.e. the number of items in the local cache is always equal to the node degree. Local diffuser uses algorithm 1 with communication complexity $O(|E(G)|)$.

Algorithm 1 Local diffuser

1: **for all** Edge $e = (x, y)$ *of* G_B **do**
2: set $D'_x \leftarrow D'_x \cup \{y\}$ {adds y's descriptor to the cache of x}
3: set $D'_y \leftarrow D'_y \cup \{x\}$ {adds x's descriptor to the cache of y}
4: **end for**

4.4.1.2 Exchange Diffuser

Exchange Diffuser simulates the rumor spreading based resource discovery. It exchanges cache's content of each node with one of its direct neighbor that is chosen at random with equal probabilities. The algorithm is shown in 2. The communication complexity of Exchange diffuser using PUSH/PULL protocol is $O(|V(G)|)$.

Algorithm 2 Exchange diffuser

1: **for all** node $x \in V(G_B)$ **do**
2: let y be a neighbor node chosen at random
3: set $D'_x \leftarrow D'_x \cup D_y$ {PULL protocol}
4: set $D'_y \leftarrow D'_y \cup D_x$ {PUSH protocol}
5: **end for**

4.4.1.3 Transferor diffuser

Transferor diffuser is a network agent that manages and controls mobile agent transferring between nodes. The behavior of the transferor diffuser is shown in algorithm 3. Its communication complexity is $O(M)$, where M is the number of active mobile agents in the network. At the beginning of the simulation, each node creates one mobile agent and dispatches it to the network. Therefore, the total number of mobile agents M is equal to the number of nodes $\mid V(G) \mid$. It picks a mobile

Algorithm 3 Transferor diffuser

1: **for all** node $n \in V(G_B)$ **do**
2: **for all** mobile agent $m \in A'_n$ **do**
3: let Node $d \leftarrow m.destination()$
4: $A_d \leftarrow A_d \cup \{m\}$
5: **end for**
6: **end for**

agent from the leaving buffer A' of a node and drops it in the arrival buffer A of another node that is chosen as destination. The mobile agent chooses by itself its next destination from one of direct neighbors of the welcomed node, see 4.4.2.2.

4.4.1.4 Migrator

Migrator is a simulator related agent that coordinates job movement process between initiating node from one side and selected node(s) from another side. Migrator behaves according to the algorithm 4.

Depending on the number of destinations in the balancing vector grp of the each node, algorithm choose between two methods to execute. $migrate(source, destination)$ method is chosen when there is only one destination

i.e. when the grp is resulting from performing SID, RID, or HLM strategy. Performing Sandpile strategy produces the balancing vector grp with one source and two destinations, therefore migrator agent uses method $migrate(source, dest1, dest2)$.

Algorithm 5 simulates the job migration for Sandpile strategy. According to Laredo et. al.[Laredo 2014], the migration between the source and destination nodes is made when the number of jobs at the source node is larger than the sum of number of jobs at the destination nodes. The number of jobs to be migrated from source node equals the number of jobs minus the average number of jobs at the all nodes. for each jobs migration the node with the lowest jobs count is selected as the destination. Depending on the actual state of the destination node, a job is selected from the source node, which has to be compatible and its migration is profitable. i.e its waiting time at source node is greater than the queue length of the destination node plus the required transfer duration, see 7. After a jobs is migrated using the algorithm 8, the number of jobs at the destination node is incremented. Finally, If the $reinforcement$ parameter is set, the source node updates its cache with the actual state of the two selected destinations.

Algorithm 6 is used by migrated when SID, RID, or HLM strategy is selected for doing load balancing. First, a suitable job is selected in same way that explained in the 5. Then, the selected job is migrated to the destination node. Again, if the $reinforcement$ parameter is set, the source node updates its cache with the actual state of the selected destination for SID and of the source node for RID or HLM, the destination node updates its cache with the actual state of the selected source node.

Selection algorithm 7 is executed by the source node. It receives a descriptor that represents the actual state (not the one used by load balancing strategy) of the selected destination. Using provided information, the minimum bandwidth between the source and the destination is determined. Iterating over all jobs in the waiting queue, if the required architecture of the job in the hand is compatible with the architecture of the destination node and its queue time in the current node is larger than the summation of the total queue time of destination node plus the transfer time then the it is added to the list of eligible jobs. When the resulting list is

Algorithm 4 Migrator

```
1: for all node n ∈ V(G_B) do
2:    grp=n.getgroup()
3:    if grp ≠ φ then
4:       if grp.length > 2 then
5:          migrate(grp[0], grp[1], grp[2])
6:       else
7:          migrate(grp[0], grp[1])
8:       end if
9:    end if
10: end for
```

Algorithm 5 migrate(source,destination1, destination2)

```
1: l_0 = count(jobs, source)
2: l_1 = count(jobs, destination1)
3: l_2 = count(jobs, destination2)
4: if l_0 > l_1 + l_2 then {Laredo et.al. criteria}
5:    x = (l_0 + l_1 + l_2)/3
6:    jobs = ⌈l_0 − x⌉
7:    for i = 0; i < jobs; i++ do
8:       if l_1 == min(l_1, l_2) then
9:          job ← source.select(destination1){see 7
10:         if job ≠ null then
11:            migrate(source, destination1, job)
12:            l_1++
13:         end if
14:      else
15:         job ← source.select(destination2)
16:         if job ≠ null then
17:            migrate(source, destination2, job)}
18:            l_2++
19:         end if
20:      end if
21:   end for
22: end if
23: if (reinforcement = true) then
24:    source.update(destination1)
25:    source.update(destination2)
26: end if
```

Algorithm 6 migrate(source, destination)

```
1: job = source.select(destination);
2: if job ≠ null then
3:    migrate(src, dest, job)
4: end if
5: if (reinforcement = true) then
6:    if (strategy = SID) then
7:       source.update(destination)
8:    else
9:       destination.update(source)
10:   end if
11: end if
```

not empty, and according to the used job selection policy, one job is selected for

Algorithm 7 select(destination)

1: $bandwidth \leftarrow min(bandwidth_{source}, bandwidth_{destination})$
2: $eligible = new\ List()$
3: $Qlength = QueueLength(destination)$
4: **for all** $(job : Queue)$ **do**
5: **if** $(isCompatible(job, destination))$ **then**
6: $transTime = \lceil Size(job)/bandwidth \rceil$
7: **if** $(QueueTime(job) > (Qlength + transTime))$ **then**
8: $add(job, eligibles)$
9: **end if**
10: **end if**
11: **end for**
12: **if** $eligible \neq \phi$ **then**
13: $policy = Balancing.getSelectionPolicy()$
14: **if** $policy = earliest$ **then**
15: $job = min(eligibles, SubmittionTime)$
16: **else if** $policy = latest$ **then**
17: $job = max(eligibles, SubmittionTime)$
18: **else if** $policy = shortest$ **then**
19: $job = min(eligibles, runTime)$
20: **else**$policy = smallest$
21: $job = min(eligibles, jobSize)$
22: **end if**
23: $return(job)$
24: **end if**
25: $return(null)$

migration. As explained in see 3.2.1, the following selections are possible: the job with earliest submission time, the job with latest submission time, the jobs with shortest service time, or the jobs with smallest jobs size.

Algorithm 8 move(source, destination,job)

1: **if** $job.isReady()$ **then**
2: $bandwidth \leftarrow min(bandwidth_{source}, bandwidth_{destination)})$
3: $trans_{time} \leftarrow size_{job}/bandwidth$
4: $job.setArriveAfter(trans_{time})$
5: $destination.J_{in} \cup job$ {dropping the job at receiving buffer}
6: $activeness_{destination} + +$
7: $source.remove(job)$
8: $activeness_{source} - -$
9: $updateIndicator(source, destination))$ {updates migration graph, see 9}
10: **end if**

The final phase of the load balancing process is the movement of the selected job from the source node to destination node. Algorithm 8 explains the implemented steps. The minimum bandwidth between the two nodes is determined since the time required to transfer the jobs is affected by the bandwidth limitations. The transfer time is calculated on the size (in Mega Byte) and the bandwidth. The job is expected to be ready at the receiving buffer of destination node after one or several cycles. The process is always considered successful and definitive. Hence, the job is removed from the waiting queue of the source node. For system behavior study, the algorithm updates the indicator value of the arc from the source to destination node in the migration graph G_m according to the algorithm explained in 9. Algorithm 9 update the structure or the information stored in the migration

Algorithm 9 updateIndicator(source,destination)

1: **if** $(source, destination) \notin A(G_m)$ **then** {first migration between the two nodes}
2: $A(G_m) = A(G_m) \cup (source, destination)$ {add new arc}
3: $weight_{(source,destination)} = 1$ {set the weight value}
4: $pheromone_{(source,destination)} = 0.5$ {set the pheromone value}
5: **else**
6: $weight_{(source,destination)} + +$ {increments the weight value}
7: $T = now() - lastupdate(source, destination)$;{compute the duration since last update}
8: **if** $T > 0$ **then**
9: $evaporation = (1/T)$
10: **else**
11: $evaporation = 1$ {no evaporation}
12: **end if**
13: $delta = 0.5$; {deposing amount for each migration}
14: $pheromone_{(source,destination)} = pheromone_{(source,destination)} * evaporation + delta$;
15: $lastUpdate_{(source,destination)} = now()$
16: **end if**

graph for each migration event. An arc is added for the first migration is made from the specified source to the specified destination and the initial values of weight and pheromone are set. For each next migration, the weight value is incremented and a new pheromone is deposited to the remaining concentrate that is evaporated relatively to last deposing time.

4.4.2 Node agents

Node agents simulate the model related agents that are working locally on node level. Node agents are grouped in a set. This set installs an agent only when it is a part of the required configuration. The *scheduler* is the core of any node. The two

other agents are necessary for the cooperation between nodes: *Promulgator* and *Balancer*. Below we explain the algorithm of each of node agent.

4.4.2.1 Promulgator

The promulgator agent simulates the activities that of the information manager agent in the model, see 3.1.2. Promulgator receives dropped information or mobile agent from input buffer or port, execute mobile agent and update local knowledge. It uses one of three optional techniques to deliver information to node. It works according to the algorithm 10.

Algorithm 10 Promulgator

Require: preference $\in$ (*Recentness*, *Activeness*)
1: **for all** agent $a \in A$ **do** {if mobile agent based method is used}
2: delete(a, A)
3: run(a)
4: put(a, A')
5: **end for**
6: $D' \leftarrow (D' \cup D)$ {merges new information with old cache}
7: $checkConsistancy(D', preference)$ {drops invalid items from the cache}
8: $D \leftarrow D'$ {move buffer contents to the main cache}
9: $D' \leftarrow \phi$ {clear buffer}
10: $SPAWN(cycle == 0)$ {create and dispatches a new mobile agent}

The promulgator agent first checks arrival buffer for mobile agent, if any, moves them one by one to an execution box, puts leaving mobile agents in the departure buffer. Then, it merges existing information in the main cache with new information in collecting buffer (all methods deposit collected information in this buffer), checks for information consistency using algorithm 12, and finally moves information to main cache.

4.4.2.2 Mobile agent

Mobile agent simulates the activity of the mobile agent in the model, see 3.1.2. At first cycle, Promulgator agent spawns and dispatches a mobile agent (when used) to the network. A mobile agent moves between nodes like a bee. It departs from one node with updated information, then lands at another node, exchanges information, and chose new destination. They are transferred by global agent transferor, see 3. A mobile agent behaves according to algorithm 11. A mobile agent carries only recent information in spread them when moving from one node to another. Thus, it always filters information by recentness preference using algorithm 12.

Algorithm 12 maintains collected information before moving them to the main cache to be used by the load balancing process. Each descriptor in the given buffer is validated through comparing its age with *ttl* parameter. *ttl* is the maximum

Algorithm 11 Mobile agent

let D_m be the cache of mobile agent
let x be the current node
set $D'_x \leftarrow D'_x \cup D_m$
set $D_m \leftarrow D_m \cup D_x$
$checkConsistency(D_m, recentness)$
let y be a neighbor node of x selected at random
set the next destination to y

Algorithm 12 checkConstancy(D, preference)

for all element $d \in D$ **do**
  if $age(d) > ttl$ or (isComplex and $architecture(d) \notin C$) **then**
    delete(d) {drop any outdated or incompatible information}
  end if
end for
$sort(D, preference)$
while $size(D) > K_D$ **do**
  $delete(lastIndex)$
end while

Time-To-Live value set as one of the experiment parameters. If information of none compatible nodes is also deleted when this option is set. Remaining descriptors are sorted ascending by their recentness or ascending/descending (according to sender or receiver initiation policy respectively) by their activeness. The sorting option is also set in the global configuration of the simulator. Items at the end of the list are removed to have remaining the number equals to the main cache size.

4.4.2.3 Scheduler

The scheduler agent simulates the behavior of the scheduler agent in the model, see 3.1.2. It performs three mean tasks: scheduling arriving jobs in waiting queue, releasing resources that were used by finished jobs, and allocating available resource for next job in the queue. The three tasks are explained in the algorithm 13. Algorithm 13 explains how the local resources are handled. First, the buffer of job arrival is scanned for new jobs. This buffer contains jobs that are submitted directly to the node or these who have been migrated by load balancing process. A job is considered ready for scheduling if its arrival time is greater or equal current time. A job is considered uploading in this buffer if its arrival time does not reach yet. A ready job is scheduled only if its requirements are compatible with the node resources. Otherwise it is put aside in the forwarding queue Q'. Jobs Q' are either dropped or forwarded to a compatible node according to used configuration. The scheduler agent then checks the status of the currently processing jobs. If the processing time of a job has been completed, scheduler moves it from this list

Algorithm 13 Scheduler

```
1: E be the list of jobs being executed.
2: F be the list of finished jobs.
3: for all job ∈ J′ do {Task one: schedule arriving jobs}
4:   if Arrival(job) ≥ now then
5:     if isCompatible(job) then
6:       put job at proper position in Q
7:     else
8:       put job in Q′
9:     end if
10:  end if
11: end for
12: for all job ∈ E do {Task two: release resources}
13:   if now ≥ (startTime(job) + runTime(job)) then
14:     set endTime_job ← now()
15:     put job in F
16:     remove job from E
17:     release allocated resources
18:   end if
19: end for
20: job ← Queue.head() {Task three: allocate resources}
21: if isFit(job, resources) then
22:   allocate required resources for job
23:   put job in E
24:   set startTime_job ← now
25:   Queue.remove(job)
26: end if
```

to completing jobs list to release used resources. Finally, next job in the queue using FCFS queuing policy is moved to processing list when the available resources become sufficient for its process. Finally, the job is removed from waiting queue.

4.4.2.4 Balancer

The balancer agent simulates the behavior of the Balancer agent in the model. It assesses and compares the load status of its node with information in the local cache. It initiates load balancing dialog when imbalance is detected. According to load balancing policies, it determines the source and the destination(s) to migrate jobs between eligible nodes. Balancer agent behavior is explained in algorithm 14.

The selection of both the load balancing strategy and α is made in the global configuration. Algorithm 14 computes the load of the node and the mean load of known nodes in the local cache D. The underloaded T_{under} and overloaded T_{over} thresholds are calculated depending on the value of α. Small α makes the two thresholds become near the mean load. The node state is determined weather if it

Algorithm 14 balancer

Require: $strategy \in (SID, RID, SANDPILE, HLM)$
Require: $\alpha \leftarrow\in (0, 0.5)$
1: Let L_n be the load of current node n.
2: Let $\overline{L} \leftarrow mean(|D|)$.
3: $T_{under} \leftarrow \overline{L} \times (1 + \alpha)$.
4: $T_{over} \leftarrow \overline{L} \times (1 - \alpha)$.
5: **if** $L_n > T_{over}$ **then**
6: $state \leftarrow$ overloaded
7: **else**
8: **if** $L_n < T_{under}$ **then**
9: $state \leftarrow$ underloaded
10: **end if**
11: **end if**
12: let $grp \leftarrow Null$
13: **if** $strategy == SID$ **then**
14: **if** $state == overloaded$ **then**
15: randomly select destination $\in \{u \in D : l_u < L_{under}\}$
16: let $grp \leftarrow [this, destination]$
17: **end if**
18: **else if** $strategy == RID$ **then**
19: **if** $state == underloaded$ **then**
20: randomly select source $\in \{v \in D : l_v > L_{over}\}$
21: let $grp \leftarrow [destination, this]$
22: **end if**
23: **else if** $strategy == SANDPILE$ **then**
24: randomly select two destinations $\in \{u, v \in D : l_u < L_{under}\}$
25: **if** $count(this) > count(destination1) + count(destination2)$ **then**
26: let $grp \leftarrow [this, destination1, destination1]$
27: **end if**
28: **else**$strategy == HLM$
29: **if** $state == underloaded$ **then**
30: source $\leftarrow max|D|$
31: let $grp \leftarrow [this, destination1]$
32: **end if**
33: **end if**
34: return grp;

is overloaded, intermediate, or underloaded by comparing its load to the thresholds. The *grp* vector is cleared from selection in previous cycles (if any). According to used load balancing strategy, the source and the destination node(s) are specified and stored in the *grp* vector. If the strategy is SID and the node is overloaded then a destination node is chosen at random that should has load less than T_{under} and

the node is the source. If the strategy is RID and the node is underloaded then a source node is chosen at random that should has load greater than T_{over} and the node is the destination. If the strategy is Sandpile and the node is overloaded then two destinations nodes are chosen at random and the node is the source. Finally, if the strategy is HLM then the maximum loaded node is chosen as source node and the node is the destination. The algorithm returns *grp* vector to be used by the network agent the migrator.

4.5 Workload implementation

A job is a computer program. Job attributes are organized in a class called *Job* descriptor. A job has the following attributes. Note: The first six attributes are data and the last five attributes are defined for evaluation purpose.

- Id: an identifier used to refer to the job. It created using random string plus submit time value.
- $nodeId$: the node the job is submitted to.
- $runTime$: the processing time of a job measured by cycles.
- $Arch$: a job is executed either on any architecture or it may require a specific architecture, application, resource etc.
- $Size$: the size of job file(s) measured in byte. This property is used for computing migration cost.
- $submitTime$:The time when the job arrives in the system.
- $startTime$: the time when the job starts its execution.
- $endTime$: the time when job finishes its execution.
- $status$: the current status of the job: 0 waiting, 1 executing, and 2 finished.
- $executerId$:The identification of the node that executes the job.
- $moveCounter$: the number of migrations has been made for the job.

Instances of workload are either synthetic or real. Synthetic ones are generated according to model of the literature, for example[Lublin 2003] and [Li 2005]. Real instances are obtained from specific production systems as trace files downloaded from sites like [PWA] and [GWA], see 5.1.3 for details. The simulator has a utility class called *Dispatcher* that is able to provide nodes with jobs from any source specified by the setting.

4.6 Summary

This section summarizes the steps performed by the simulator during one cycle. Jobs are submitted to a node in the network. A node receives submitted jobs in a buffer. Scheduler inspects jobs in buffer to see if their requirements are met locally. Accepted jobs are moved to waiting queue. Then, they are processed. Diffuser picks information from the output buffer of each node and drops them into input buffer of destination node. Local promulgator collects information from input buffer, sorts them by preference order, checks them for their consistency, and puts the remaining items into main cache. It delivers information to other nodes according to chosen scheme by dropping them in the output buffer. Balancer agent compares the load of nodes in descriptor table with the load of its node. According to initiation policy, it selects a source or destination(s) nodes for possible migration process. Balancer asks selected node for its current state. According to preferences, it updates its descriptor table with new up-to-date information and sees if the migration is still profitable. It selects (asks source node to select) a suitable job for migration to the destination node. If a job is selected, it is dropped into input buffer of destination node. Each node updates its activeness indicator. Migrator coordinates migration process. A scheduler agent collects migrated jobs together with submitted jobs and puts them into waiting queue. System parts interact as depicted in figure 4.2.

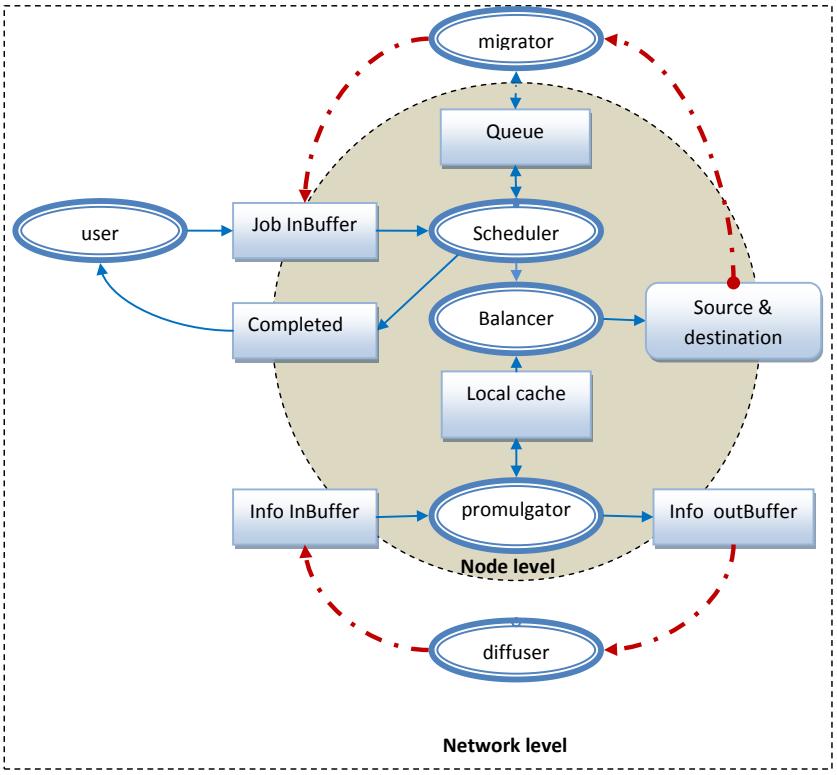


Figure 4.2: Multi-level Agent interaction

Agent takes decisions using current state of some nodes as parameter. A state of a node is a function to the input and results of historical decisions. As a result, agent behavior influences the states of some nodes or whole system when the network is connected.

Simulator uses the cycle as a discrete time notation. Simulator runs for a duration of time specified at the global configuration of the run. During this duration jobs are submitted to the nodes. After the last cycle, no new job is submitted to the nodes. However, the run is continuing until the processing of all remaining jobs is completed, see 5.2.

Chapter 5

RESULTS AND DISCUSSION

Contents

5.1 Parameters setting 66
5.1.1 Node types 66
5.1.2 Network types 67
5.1.3 Workload types 71
5.2 Simulation results 73
5.2.1 No migration 74
5.2.2 Load balancing with local scheme resource discovery 74
5.2.3 Load balancing with global scheme resource discovery 83
5.2.4 The impact of resource discovery method 93
5.2.5 Structure of the overlay network 94
5.2.6 Improving information management 102
5.2.7 Heterogeneous nodes and bandwidth 105
5.2.8 Analyzing of migration graphs 109
5.3 Discussion 114
5.3.1 Performance of the system 114
5.3.2 Structure of the overlay network 115
5.4 Conclusion 116

Discrete event simulation has been used to evaluate the performance of the considered model of distributed computing system. Many experiments have been conducted to analyze the global behavior of the system according to different policies for resource discovery and load balancing that are presented in chapter 3. The simulator that is presented in chapter 4 is used for carrying the experiments. In this chapter, the configurations of the tested model are described first, then, obtained results are presented and discussed.

A very large number of parameters and methods are investigated to study the stability conditions of DCS. Figure 5.1 show the relation between any two parameters which forms a graph. The almost-tree is rooted by the DCS, the subject of the study. The shown hierarchy focuses on the sort each parameter and how it is related to other parameters.

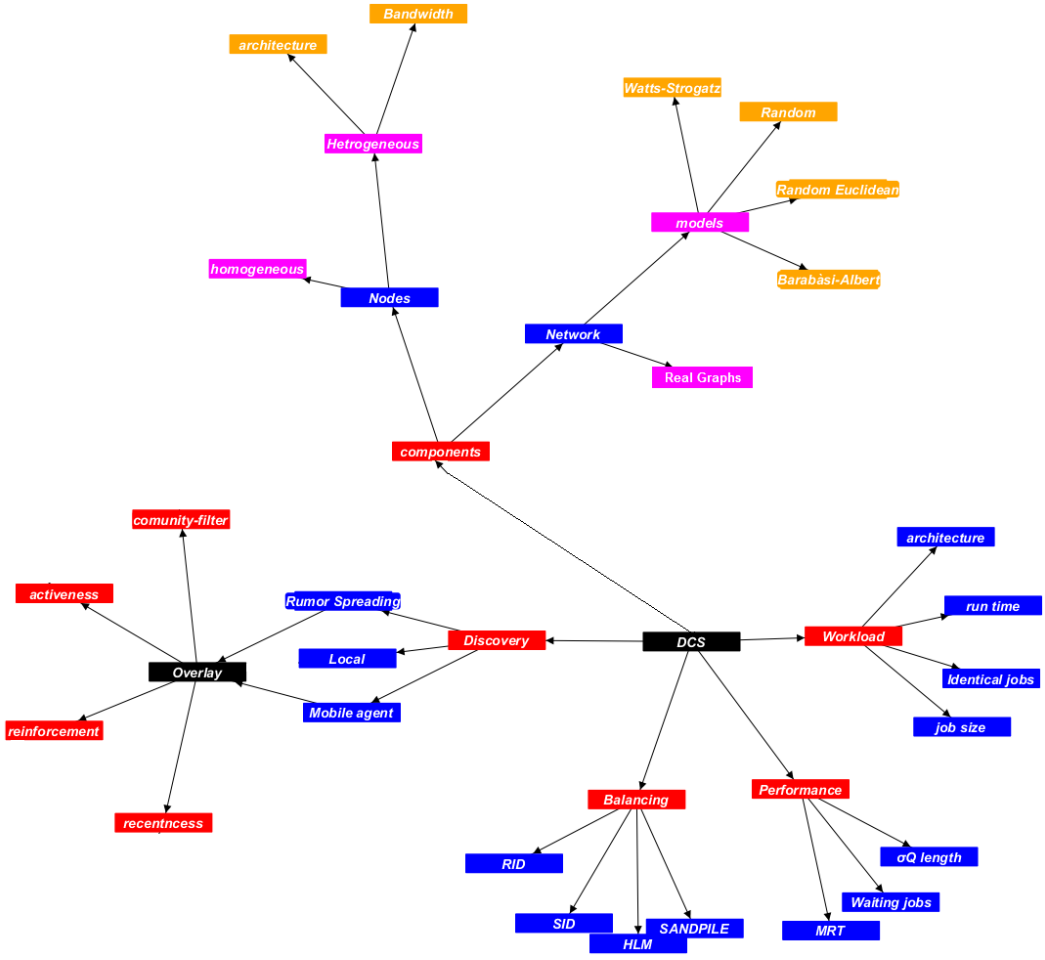


Figure 5.1: Overview of studied elements in a DCS

5.1 Parameters setting

The model behaves according to its initial parameters and the characteristics of the workload, which is the input. The details of tested configurations for both nodes and networks are given below.

5.1.1 Node types

In literature, either homogeneous or heterogeneous nodes are considered. For example, in [Laredo 2014], a group of experiments assumes a scenario in which 256 sandpile agents take control over a homogeneous architecture of $q = 256$ nodes where $\forall i, j : p_i = 1$ and $C_{i,j} = \infty$, i.e. every node computes $1\frac{instruction}{cycle}$ and migrations are instantaneous. In [Acker 2007], all the experiments use a grid of 20 nodes. A Grid network having a number of hosts equal to 2500 is considered in [Forestiero 2007]. A real network consisted of $8,584$ nodes and $19,439$ undirected and unweighted links has been used in [Fukuda 2007a]. A node corresponds to an

Table 5.1: Node Configuration types

Node type / Feature	Homogeneous	Heterogeneous	
	H	A	B
Architecture	0	$\sim U[0-4]$	0
Bandwidth	1	1	$\sim U[1-4]$

autonomous system (AS), which is either a single network or group of networks that is managed by one or more network administrator that is belonging administratively to a single institute like a university or a business enterprise [Hawkinson 1996]. And artificial one with the number of nodes was set to 10000 and the number of links to 20000. A network with $10,000$ nodes in [Shen 2014]

In [Acker 2007], each node has a 100 megabit per second link to a central switch. Network latency is the sum of all queuing and propagation delays along a path, but does not include the transmission delay. Authors consider a latency of 0.1 milliseconds (ms) to be a low value, and a 1 ms value to be high. These numbers are reasonable approximations of latencies on a lightly loaded and somewhat congested 100 megabit per second networks. Bandwidth is not considered in these studies [Laredo 2014], [Fukuda 2007a].

In our experiments, the two types of nodes are considered. Homogeneity means all nodes have same architecture and bandwidth for Internet connection. Heterogeneous nodes vary either in the architecture or in their bandwidth, see table 5.1. Node type H is the simplest configuration where all nodes have the same characteristics. In type A, nodes vary in their architecture. Five types of architecture are distributed uniformly between nodes. A system with different architecture needs more efforts on the matching jobs to suitable node. A job may have to wait more than usual, when the cache of the initial node does not contain a matching node during a long duration. Type B assigns bandwidth for nodes uniformly between 1 and 4, all nodes have the same architecture. No latency is considered, see 3.3.3. Each type of the presented configurations is tested with the different network structure. The selected models of networks are presented in the following section.

5.1.2 Network types

Nodes start communication over some "initial" network that is called the underlay network. Underlay network is represented by a graph (directed or undirected). Graph instances are obtained from two sources: real life networks and generated graphs. Real life graphs (called real graphs in the following) are snapshots from the Internet that are taken at specific times where a node may correspond to an Autonomous System AS or a host in a Peer-to-Peer network. An AS is either a single network or group of networks that is managed by one or more network administrator, and that belongs to a single institute like a university or a business enterprise [Hawkinson 1996]. These AS nodes are connected by border-gateway routers (that use Border-Gateway-Protocol BGP) creating a sub-network of Internet

that is referred by AS network. Using tracing techniques, a graph that models the network at a given time is created. Some known web sites like CAIDA [CADIA] or SNAP [Leskovec 2014] offer copies of these graphs for analyzing purposes. Note that CAIDA itself is a large network of AS systems. Finally, generated graphs are obtained using some of theoretical models that are explained in 2.4.

A Grid network of 2500 hosts is considered in the work of [Forestiero 2007]. Hosts are linked through P2P interconnections, and each host is connected to 4 peer on average. Each node has link to a central switch in [Acker 2007] and all the experiments use a grid of 20 nodes. Two types of network topology are used in [Fukuda 2007a]: real-life network taken from CAIDA [CADIA] with undirected and unweighted links with average degree equals 2.3. And, similar simulations are performed using Generalized Barabási-Albert-generated topology. In [Laredo 2014] Ring and Small-world model is used. In [Shen 2014], every node is deployed in such a way that the node links to about 10 neighbors.

Five instances of real graphs are used. An instance of a graph is chosen from the specified set arbitrary. However, graph with different sizes were chosen. A description for each real graph is given below:

Autonomous systems AS-733: The dataset was collected from University of Oregon Route Views Project - Online data and reports. The dataset contains 733 daily instances which span an interval of 785 days from November 8 1997 to January 2 2000. The dataset was used in [Leskovec 2005]. Instance of Dec 13 1999 is chosen. Its order is 2071 nodes.

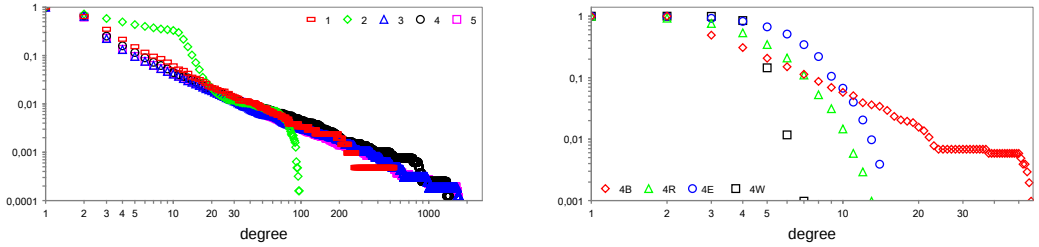
Gnutella peer-to-peer network, August 8 2002: A sequence of snapshots of the Gnutella peer-to-peer file sharing network from August 2002. Nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts. There are total of 9 snapshots of Gnutella network collected during August 2002. One snapshot of order 6299 nodes has been selected.

CAIDA AS Relationships Datasets: The dataset contains 122 CAIDA AS graphs that are collected from January 2004 to November 2007. Each file contains a full AS graph derived from a set of RouteViews BGP (Border gateway protocol) table snapshots. Snapshots of October 05, 2004, September 17, 2007, and November 05 2007 have been chosen[CADIA2007]. Their orders are 16301, 8020, 26475 nodes respectively.

For simplicity issue numbers 1, 2, 3, 4, and 5 are used to refer to real graphs ordered as they were presented. Figure 5.2a shows their Inverse Cumulative Degree Distribution $ICDD(k)$, see 2.4.1. The figure shows that graph 1,3,4, and 5 have a power-low degree distribution, which is the important feature of AS graphs. Graph 2 shows a different degree distribution since it is a snapshot of file exchange of a Peer-to-Peer network. Indeed, it is a snapshot of an overlay network in which most peers approximately have same rights of number of connections, which is limited only by their connection bandwidth. Hence, it differs from AS graphs by the $\sigma(G)$ and $\overline{C}$, see figure 5.3.

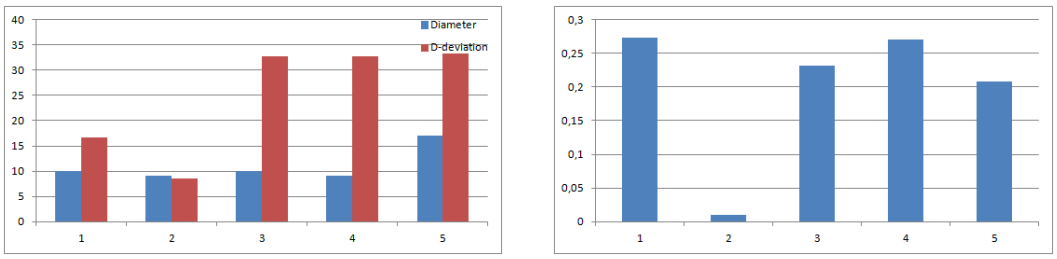
Four sets of instances have been prepared using GraphStream [Pigné 2008][GraphStream] generators. They use the graph models presented in

2.4, that are used in the DCS literature. All graphs are connected and have an order of 1024 nodes. From each model, six instances with different average degree have been created. Average degrees of 4, 8, 16, 24, 32, and 48 are considered. Increasing the average degree decreases the diameter of a graph that is generated of same model.



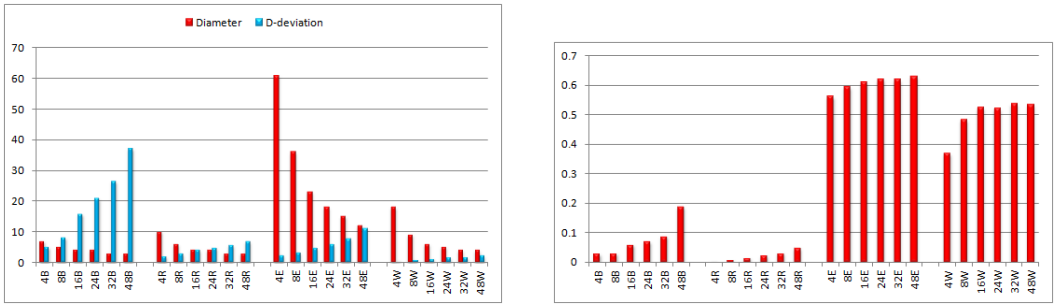
(a) Real graphs (b) Generated graphs of $\overline{D} = 4$

Figure 5.2: Inverse cumulative degree distribution of real and generated graphs



(a) $\sigma(G)$ *and* $d(G)$ (b) $\overline{C}$

Figure 5.3: Characteristics of real graphs



(a) $\sigma(G)$ *and* $D(G)$ (b) $\overline{C}$

Figure 5.4: Characteristics of generated graphs

The average degree and the diameter of Random Euclidean graphs depend on

the threshold of the distance between points in the plane to be connected. A small threshold has to be chosen to generate a graph of small average degree. But, using such small threshold generates most of the time a disconnected one. However, after a large number of attempts, a connected graph has been generated with a smallest possible average degree which equals $5,7$.

In the remaining of this chapter, we refer to generated instances by their average degree followed by the first letter of the generator model e.g. $8B$ means an instance that has 8 average degree and was generated according to Barabàsi –Albert model.

Figure 5.2b show $ICDD(k)$ of generated graphs of $\overline{deg}(G) = 4$. Other characteristics of generated graphs are shown graphically in figure 5.4. The two figures explain the different characteristics of graphs resulting from different models. Hence, Barabàsi-Albert and Random models have different degree distributions but same $\overline{C}$ and diameter while Random Euclidean and Watts-Strogatz have different diameters but similar $\overline{C}$ and degree distribution.

Table 5.2: Features of all graph instances

Graph Model	$d(G)$	$\Delta(G)$	$\delta(G)$	$\overline{deg}(G)$	$\sigma(G)$	$\overline{C}$
(1)as-19991213	10	524	1	4,4	16,708	0,273
(2)p2p-Gnutella08	9	97	1	6,6	8,541	0,010
(3)as-caida20040105	10	2331	1	4,0	32,756	0,232
(4)as-caida20070917	9	1452	1	4,5	32,683	0,271
(5)as-caida20071105	17	2628	1	4,0	33,374	0,208
Barabàsi –Albert	7	56	2	4	4.994	0,028
	5	95	4	8	8,13	0,03
	4	149	8	16	15,773	0,059
	4	175	12	24	20,827	0,07
	3	210	16	32	26,514	0,088
	3	258	24	48	37,139	0,188
Random graph	10	13	1	4	2,002	0,005
	6	19	1	8	2,787	0,008
	4	30	5	16	4,056	0,015
	4	38	9	24	4,748	0,023
	3	55	18	32	5,628	0,031
	3	66	29	48	6,738	0,047
Random Euclidean	61	14	1	5,7	2,395	0,562
	36	19	1	8	3,198	0,595
	23	30	1	16	4,692	0,611
	18	41	6	24	5,83	0,621
	15	53	12	32	7,76	0,62
	12	76	12	48	11,257	0,63
Watts –Strogatz	18	7	2	4	0,595	0,369
	9	12	5	8	0,837	0,484
	6	20	12	16	1,151	0,525
	5	29	19	24	1,554	0,521
	4	38	25	32	1,737	0,538
	4	57	42	48	2,16	0,535

Table 5.2 lists all instances of graphs that have been tested and their correspond-

ing features. All graphs are connected. Note that: $d(G)$ is the diameter computed as longest shortest path, $\Delta(G)$ is the maximum node degree, $\delta(G)$ is the minimum node degree, and $\overline{C}$ is the average clustering coefficient, which is the arithmetic average of the clustering coefficient of all nodes. A clustering coefficient of a node is the rate of the number of the existing edges between its neighbors to the total number of possible edges among them. see 2.4.1. $\sigma(G)$ is the deviation of node degree. The first five instances represent snapshots of real networks.

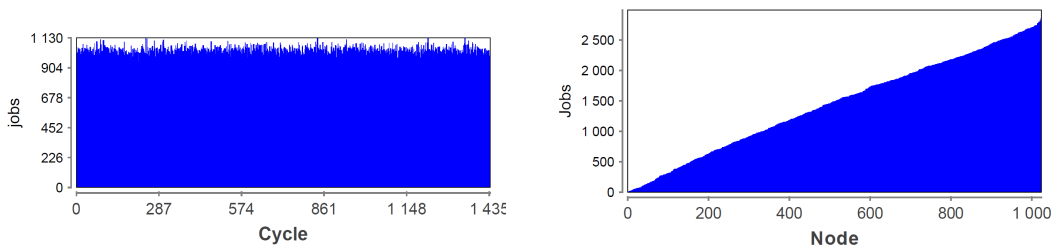
5.1.3 Workload types

Jobs, sometimes called tasks are usually prepared as sets of workload when calculating scheduling algorithms. In [Laredo 2014], the workload is composed of 256000 homogeneous tasks with a number of $n_i = 1$ instructions. The 256000 tasks arrive in a single batch (all tasks arrive at t_0, or alternatively, $\forall i : a_i = 0$) which is initially allocated in a node acting as a front-end. In [Acker 2007], in all simulation experiments, the average job size is $100,000$ bytes. Job sizes are varied uniformly from $50,000$ bytes to $150,000 bytes$. Authors claim that these sizes represent the approximate average size of a binary executable on the Linux CentOS 4 system used for the simulations. A job requires an average of 5 seconds of CPU time and 5 seconds of I/O time. The job arrival rate is 1/0.6, increasing to 1/0.5, then 1/1.25, finally, 1/0.35 seconds.

Workload instances are generated to study the behavior of a system. Four attributes of job are considered : submit time, processing time, size, and requested architecture. The impact of each attribute is investigated independently from the others.

The total number of jobs submitted during one cycle follows Poisson Binomial Distribution PBD. Each cycle, for each node v_i, n jobs are created. $n \sim$ Poisson distribution with mean parameter λ_i which is the mean arrival rate of node v_i. $\lambda_i \sim$ Uniform in range $[0, 2\lambda]$, see 3.1.4.

An instance of global arrival distribution with $\lambda = 1.0$ for 1440 cycles is shown in figure 5.5.



(a) job per cycle (b) job per node

Figure 5.5: Histogram of job arrivals during 1440 cycles

In this context, four types are defined for considering workload instance. To

compute the impact of modeled arrival rate, type *Identical* is used, where jobs differ in their submit time only, Other attributes are constant (processing time=1, size=1, and architecture=0). In remaining types, jobs vary in only one attribute. Processing time and size in type $DiffProc$ and $DiffSize$ respectively are sampled from the exponential distribution with mean value $\mu \simeq 1$. Finally, type $DiffArch$ considers an architecture that is sampled from a discrete uniform distribution in range $[0, 4]$. In last type (heterogeneous resources), some arrived jobs cannot be processed locally. They are either rejected (when migration is disabled) or forwarded to other nodes. Table 5.3 summarize the four types. In discrete simulation, a time unit is called *cycle* or *round*. The former is used here. A cycle might represent one minute, i.e. if a simulation should span one day, it has to complete at least (1440 cycles).

For a system composed of N nodes, if jobs arrive according to Poisson distribution, jobs request generic architecture, and the average job processing time is l, then, the maximum number of jobs J to be executed during a duration of T cycles is equal to:

$$J = \frac{NT}{l} \tag{5.1}$$

For example, consider a network of 1024 nodes (like most networks above), node type is H, and job type is *Identical*, such system can process a number of jobs during one day at most:

$$1474560 = \frac{1024 \times 1440}{1}$$

We must have: $\lambda < \mu$, as λ is the mean of the λ_i. Hence, $\lambda = 0.95$ is chosen.

All runs use FCFS scheduling policy. Two types of measures are needed for state monitoring and comparison. Measures that are computed as a function of time are used to monitor the system state during the simulation run. To compare between the performances of two runs, measures that are computed at the end of the run are used.

Three measures are used to evaluate the behavior of the system (i.e. the performance of load balancing) during simulation run, see 3.4: σQ, *Waiting jobs*, and MRT. σQ is the deviation of queue length from the average, which is used for measuring the performance of load balancing process. An equilibrium point is considered when all nodes have approximately same amount of load (i.e. small σQ). *Waiting jobs* is the number of waiting jobs in the system, which is used for monitoring jobs flow in the system.

Table 5.3: Job types in workload model

type reference Job requirements	*Identical*	Different		
		$DiffProc$	$DiffSize$	$DiffArch$
Arrival rate	$Poisson(\lambda_i)$, $\lambda_i \sim U[0, 2\lambda]$			
Processing time	1	$exp(\mu \simeq 1)$	1	1
Size (Mbyte)	1	1	$exp(\mu \simeq 1)$	1
Architecture	0	0	0	$\sim U[0, 4]$

The system is said to be in a steady state when the value of the measure becomes constant. MRT is the mean response time of jobs that are just completed at given time or duration. MRT may used instead of the two measures.

However, σQ and $Waiting\ jobs$ are used when the difference of MRT between two configurations is too small. Total MRT that is computed for all submitted jobs (i.e. at the end of simulation run) is used for comparing a large number of simulation runs.

5.2 Simulation results

Experiments are grouped into categories according to the specified objective. In each experiment the input, the configuration, or both are changed. The input is the workload. The configuration is the values of the parameters like: node type, underlay network graph, discovery method, load balancing strategy, $TTL(Time - To - Live)$ of information, and cache size.

Experiments are summarized as follow:

1. No migration: Each work serves receiving jobs without any information exchange and cooperation with other nodes.

2. Load balancing:

   (a) with local scheme resource discovery: a node makes load balancing using information about its direct neighbors only.

   (b) with global scheme of resource discovery: two sophisticated resource discovery methods are used to provide nodes with information about neighbors in a given range.

3. Overlay structure: studying overlay structure using global scheme resource discovery.

4. Local cache management: Feedback from old migration decisions is used to have local cache containing most required information.

5. Heterogeneous nodes: Managing system composites of nodes with different attributes through controlling local cache content.

6. Analyzing migration graphs: calculating characteristics of migration graphs resulting from interaction capturing model.

Table 5.4 lists experiments, used input and configurations. "–" means that an element or a method is not applicable. The four load balancing strategies, see 2.5.4, are tested in experiment when workload is supplied. Numbers between parenthesizes represent the size of the parameter set. The results for each group of experiments are explained in the following sections.

Table 5.4: Experiments and their configurations

Experiment groups	Workload (5)	Configuration				
		Network (29)	Node (3)	discovery (3)	TTL (9)	cache (6)
1	ALL	–	ALL	–	–	–
2.(a)	$Identical$, $DiffProc$	ALL	H	Local	–	–
2.(b)	$Identical$, $DiffProc$	ALL	H	Rumor, mobile	3	4,8,16, 24,32,48
3	–	ALL	H	Rumor, mobile	2-9	4,8,16, 24,32,48
4	$Identical$	ALL	H	Rumor	3	16
5	$DiffArch$, $DiffSize$	All	A,B	Rumor	3	16
6	selected networks from above					

5.2.1 No migration

When no resource discovery is enabled, nodes become isolated, and consequentially no load balancing can be made. Of course, no equilibrium can be reached since many jobs arrive to some nodes while other nodes receive no jobs for long intervals. A static schedule has a finishing time of two days because the maximum arrival rate is $2\lambda = 2\ jobs/cycle$ for the most weighted node.

5.2.2 Load balancing with local scheme resource discovery

This group of experiments tries to test a set of methods and environment conditions by which the system behavior is affected. Moreover, it tries to emphasize the methods and/or conditions that help system behavior to converge rapidly to a steady state. System behavior is a function of load balancing strategy, underlay network structure, resource discovery method, and some other parameters. Four load balancing strategies are tested, see 2.5.4. For an easier reading, they are briefly represented here:

1. SID: The source is the initiator. The destination is chosen at random, among underloaded nodes in the neighborhood.

2. RID: The destination is the initiator. The source is chosen at random, among overloaded nodes in the neighborhood.

3. Sandpile: The source is the initiator, the destinations are two neighbors nodes that are chosen at random when the summation of their load is less than the load of the initiator. Sandpile is also an SID but it differs in the way by which the destination nodes are selected.

4. HLM: The destination is the initiator. The source node is the maximum loaded node in its neighborhood. HLM is also an RID that chooses the source node differently.

The impact of the underlay network structure is also investigated using all instances of graphs that are listed in table 5.2. Three resource discovery paradigms are tested: local scheme, mobile agents, and rumor spreading based methods, see 3.2. $TTL = 3$ is chosen. In 5.2.5, we explain why this value is good. 4, 8, 16, 24, 32, and 48 are tested for cache size. These values are chosen for comparing between underlay and overlay network structures of same average degree. Finally two job types ($Identical$ and $DiffProc$) are tested.

5.2.2.1 Real graphs

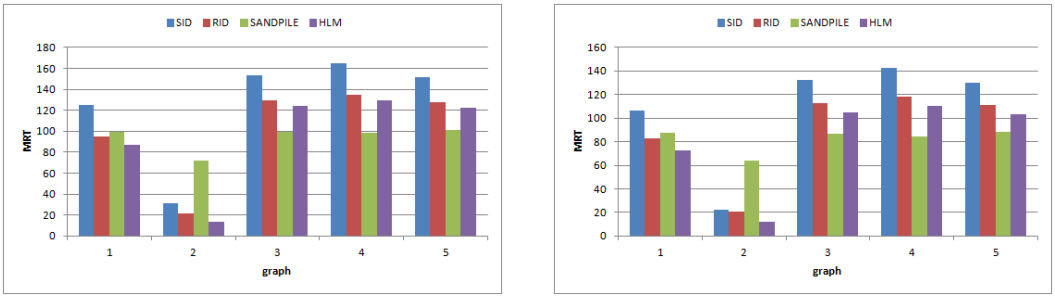
Local scheme assumes a node always knows only the state of its direct neighbors in the underlay network. Hence, overlay network has the same structure as the underlay network. It is static since cache contains the same items each cycle. Figure 5.6 shows total MRT computed for both job types. Figure 5.7 shows MRT computed for completed jobs at each cycle, on each of real graphs, and for identical job types.

The results are similar for both job types. Large and growing MRT's are noticed. Note that average degree is rather small for the five graphs. Using this scheme of resource discovery on real graphs, leaves most nodes with insufficient knowledge for developing a global solution. Therefore, an equilibrium state is not reachable using any of load balancing strategies. Nodes with high arrival rate continue accumulate many jobs in their waiting queue.

Three load balancing strategies perform the same (with small differences) in each graph. The order of the load balancing strategies are HLM, RID, SID in all graph instances. Sandpile performs different in different graphs. It is the third in graph 1, the worst of all in graph 2, and the best in graph 3,4, and 5.

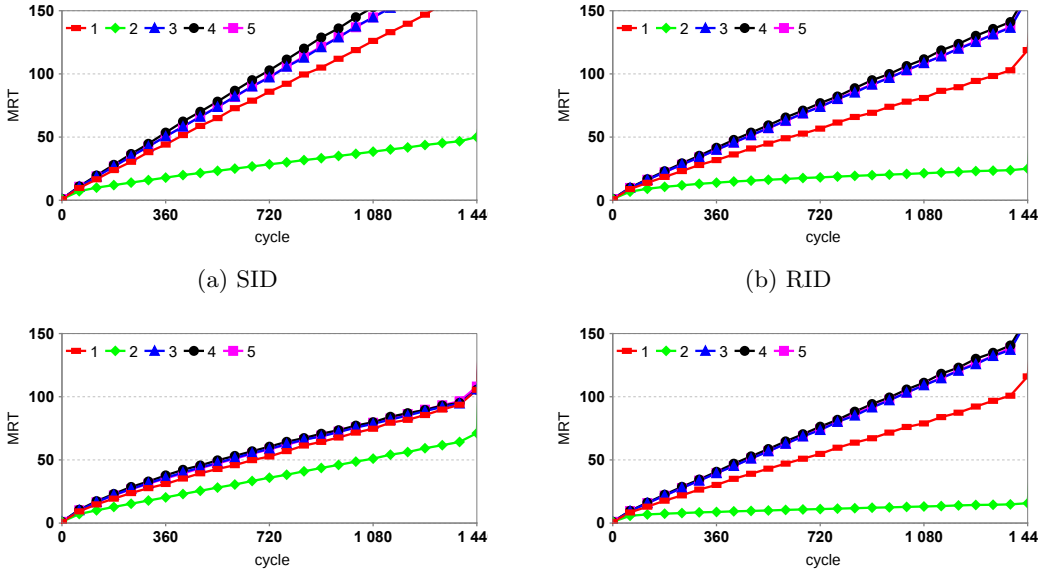
The best performance of all strategies is with graph 2 even when it has not the smallest order. Graph 2 has a small $d(G)$, smallest $\overline{C}$ and a small σD i.e it is the most regular one.

HLM and RID have relatively same performance because both are receiver initiated. Sandpile allows an overloaded node to send many jobs to two selected underloaded nodes in each migration. The other strategies permit sending or receiving only one job a time. Sandpile prefers high average clustering coefficient and high degree deviation when the network is static.



(a) *Identical* job type (b) *Diffproc* job type

Figure 5.6: Total MRT for each load balancing strategy on each real graph



(a) SID (b) RID

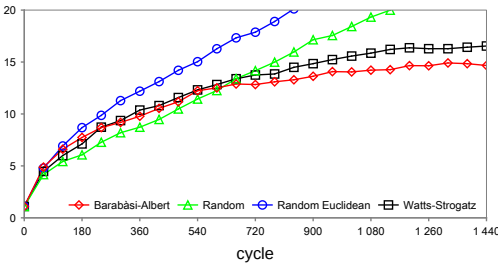
(c) SandPile (d) HLM

Figure 5.7: MRT for each load balancing strategy for Identical job type on real graphs

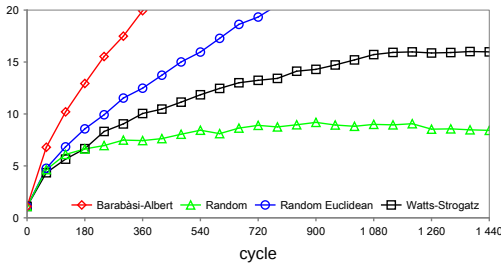
5.2.2.2 Generated graphs

The same tests are made using generated graphs. We start testing graphs from the average degree that is similar to average degree of real graphs. Two job types are used. Figures 5.8 and 5.9 show the σQ in generated graphs of $\overline{deg}(G) = 4$ using *Identical* and *DiffProcl* job type respectively.

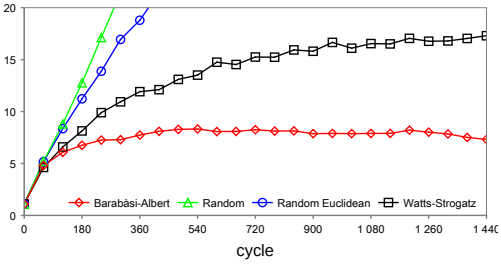
Same results for both jobs types are observed except for HLM load balancing strategy, which performs better with jobs of variable processing time in three graphs instances. HLM gets load from maximum loaded node in the neighbors, so the different between the source and destination node is highly probable to be large. Hence, the coarse grain jobs accelerate balancing process in this strategy.



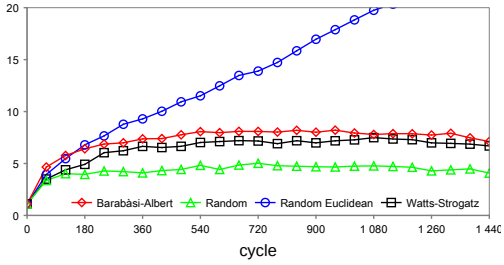
(a) SID



(b) RID

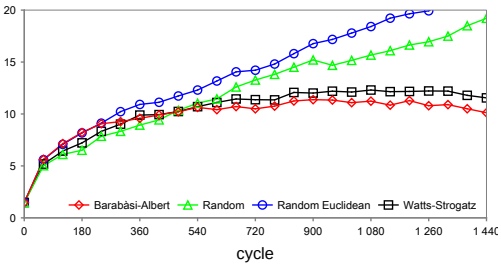


(c) SandPile

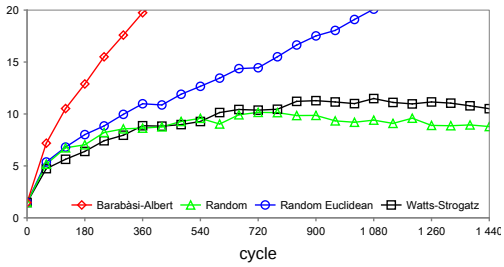


(d) HLM

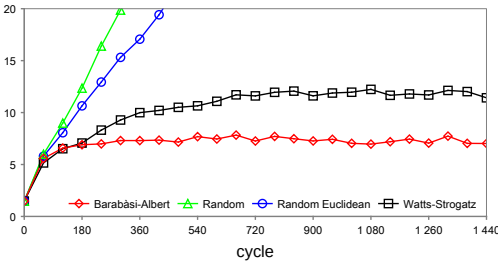
Figure 5.8: σQ in generated graphs of $\overline{deg}(G) = 4$ using Identical job type



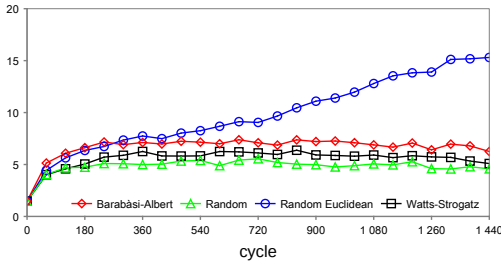
(a) SID



(b) RID



(c) SandPile



(d) HLM

Figure 5.9: σQ in generated graphs of $\overline{deg}(G) = 4$ using $DiffProcl$ job type

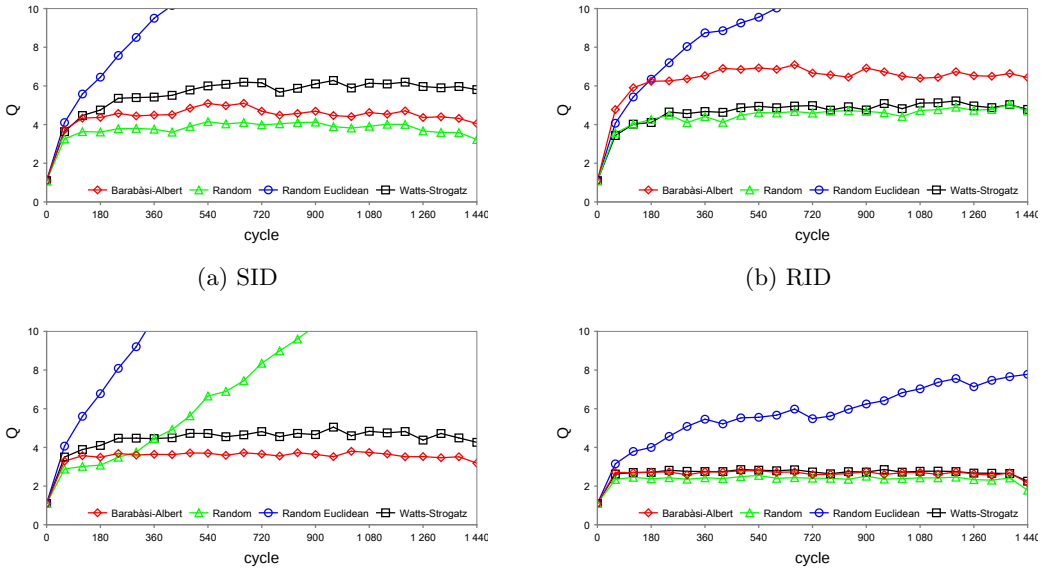
The load balancing strategies do not reach the equilibrium state for Random

Euclidean. Actually, the system does not converge to an equilibrium state using SID in all graphs. Because the number of possible destination nodes is very small e.g. two nodes in average as the average degree is four, which makes SID does maximum of two migrations in such graphs. Sandpile performs best in Barabàsi-Albert, which is the graph with most degree deviation.

RID and HLM have approximately same performance in all graph instances. However, HLM outperforms all other strategies. It produces lowest σQ. The system converges rapidly to an equilibrium state and keeps a steady state in all graph models except Random Euclidean.

Random Euclidean graph is characterized by long $d(G)$ and large $\overline{C}$. The two characteristics block load from spreading in the network using HLM. In long diameter network, a job might be migrated many times before reaching the destination node where it should be processed. Clustered network restricts the strategy to choose sources from its direct neighbors. When the cluster size (its number of nodes) is small, jobs are blocked inside these clusters. It's even truer for RID as the source is chosen randomly in the neighborhood.

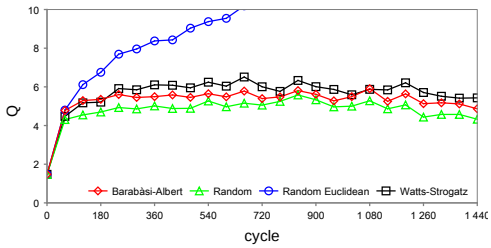
Now, we test higher average degrees to see its impact. Figures 5.10 and 5.11 show σQ for each load balancing strategy in generated graphs. Figures 5.12 and 5.13 show number of waiting jobs in the system. All these figures are for graphs having average degree 8 only. Figures 5.14 and 5.15 summarize MRT obtained in generated graph for job types *Identical* and *DiffProcl* respectively. The X-axis represents the average degree of the graph. A curve is associated a graph model. All generated graphs are included.



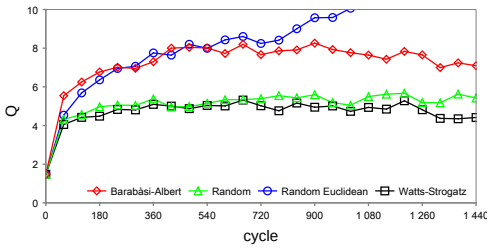
(a) SID (b) RID

(c) SandPile (d) HLM

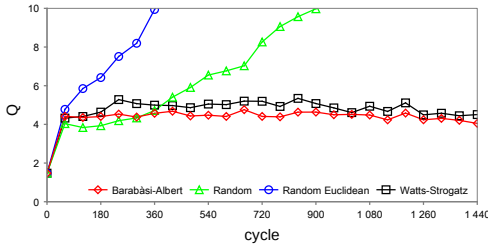
Figure 5.10: σQ in generated graphs of $\overline{deg}(G) = 8$ using Identical job type



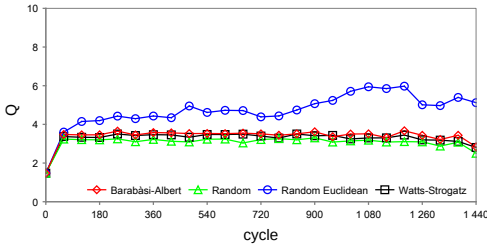
(a) SID



(b) RID

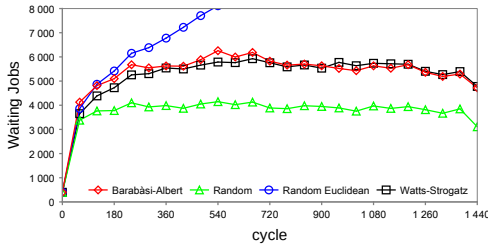


(c) SandPile

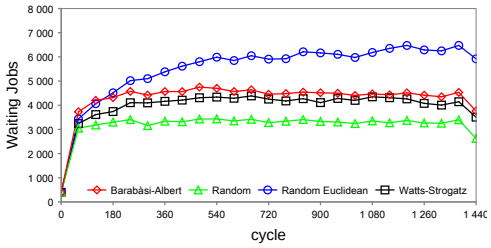


(d) HLM

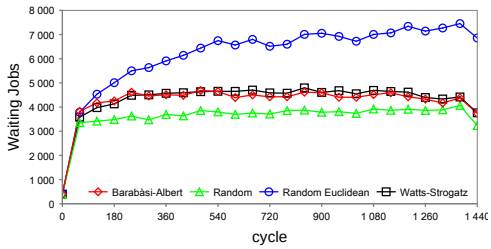
Figure 5.11: σQ in generated graphs of $\overline{deg}(G) = 8$ using $DiffProcl$ job type



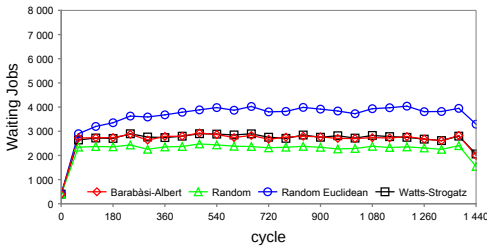
(a) SID



(b) RID

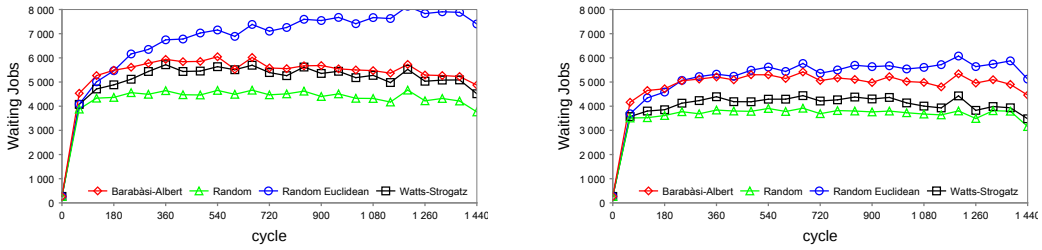


(c) SandPile



(d) HLM

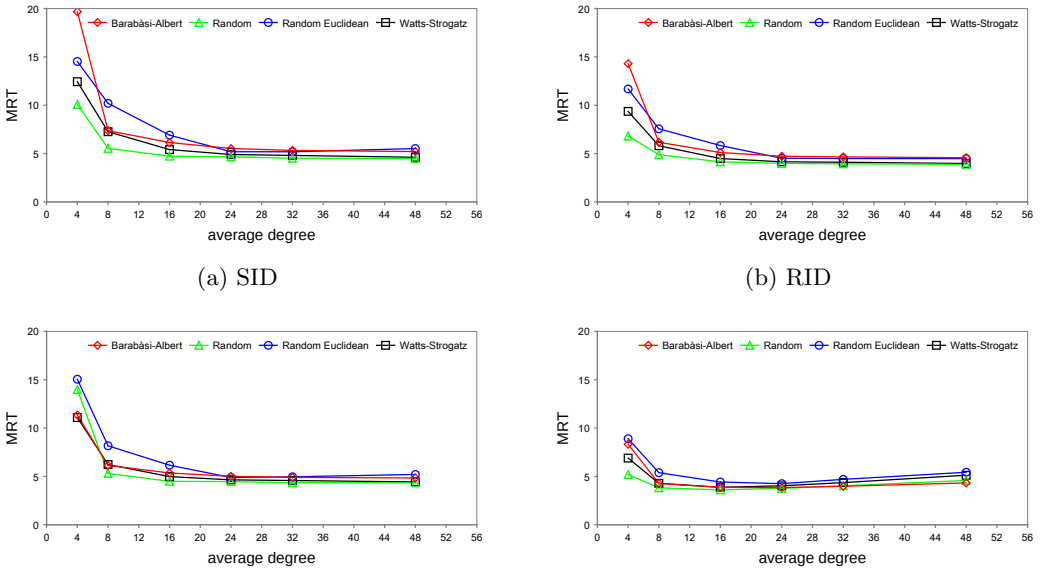
Figure 5.12: Waiting jobs in generated graphs of $\overline{deg}(G) = 8$ using Identical job type



(a) SID (b) RID

(c) SandPile (d) HLM

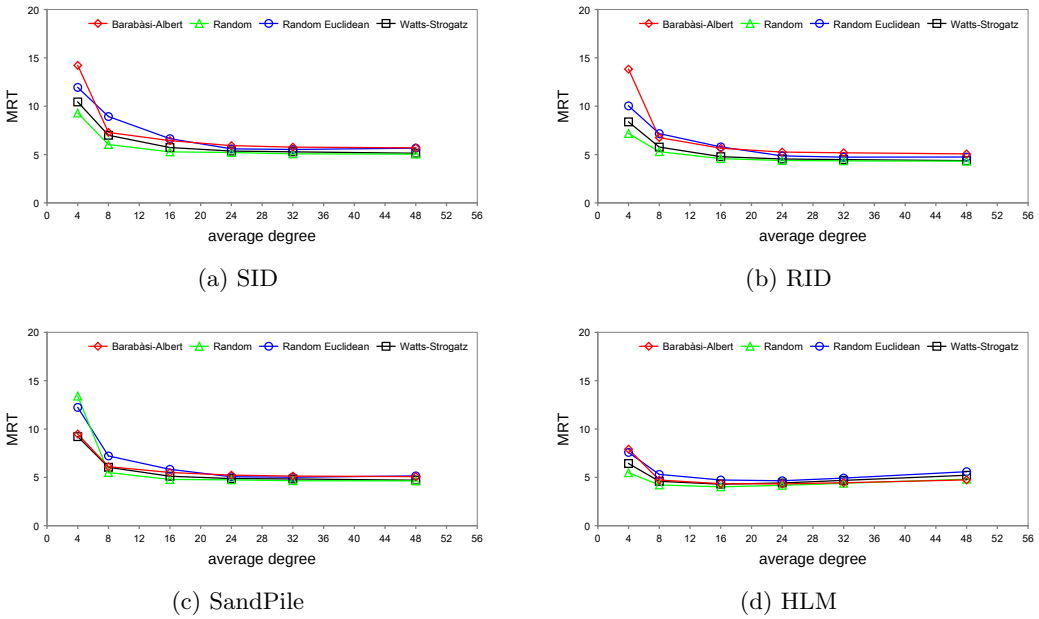
Figure 5.13: Waiting jobs in generated graphs of $\overline{deg}(G) = 8$ using $DiffProcl$ job type



(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.14: Total MRT in generated graphs at different $\overline{deg}(G)$, Identical job type



(a) SID

(b) RID

(c) SandPile

(d) HLM

Figure 5.15: Total MRT in generated graphs at different $\overline{deg}(G)$, and $DiffProc$ job type

Results are always similar for both job types *Identical* and $DiffProcl$. Receiver initiation policies (RID and HLM) outperforms Sender initiated policies (SID and Sandpile). HLM outperforms all other strategies in all graph models. System reaches an equilibrium state rapidly (approximately after cycle 150) in most network models (except Random Euclidean) using any load balancing strategy with increasing average degree.

The performance of SID, RID and Sandpile is enhanced whenever the average degree is increased. This is due to random selection of source or destination node. The performance of HLM enhances until average degree reaches 16 (the load of heavily loaded nodes decreases rapidly), then it is degraded, since the number of possible sources (maximum nodes in each neighborhood) decreases when the neighborhood size increases (as many overloaded nodes are not selected as sources). Obtained MRT is quite smaller than real graph for same run duration.

For small average degree, for all strategies, the preference of network models is ordered as Random, Watts-Strogatz, Barabàsi-Albert, and Random Euclidean, which is the same order as the order of their diameter.

Results of the applied strategy on networks of same average degree vary according to the network diameter. The smaller the diameter, the smaller the deviation.

The performance of all load balancing strategies makes the system stabilizes in all graph instances of average degree 16. A small diameter and a good average degree permit a fast movement of load between system nodes.

The uniform distribution of arrival rate on nodes makes no problem in a semi-regular graph like Random graph. However, in graphs with a high degree deviation, highest or lowest arrival rate may be assigned to critically situated nodes e.g. nodes

with very large or very small degree.

Therefore, further tests are made to compare between two types of arrival rate distributions: uniform and degree-oriented. In uniform distribution, in all experiments made so far, the arrival rate is distributed uniformly among nodes. In degree-oriented distribution[Fukuda 2007a], arrival rates are generated at random, sorted and distributed on nodes that are sorted by their degree.

Figure 5.16, 5.17 show obtained MRT using each distribution of arrival rate in real and generated graphs respectively. Generated graph are all at average degree equals 8. Other average degrees show same behavior. Note that figure 5.17 uses logarithmic Y-axis.

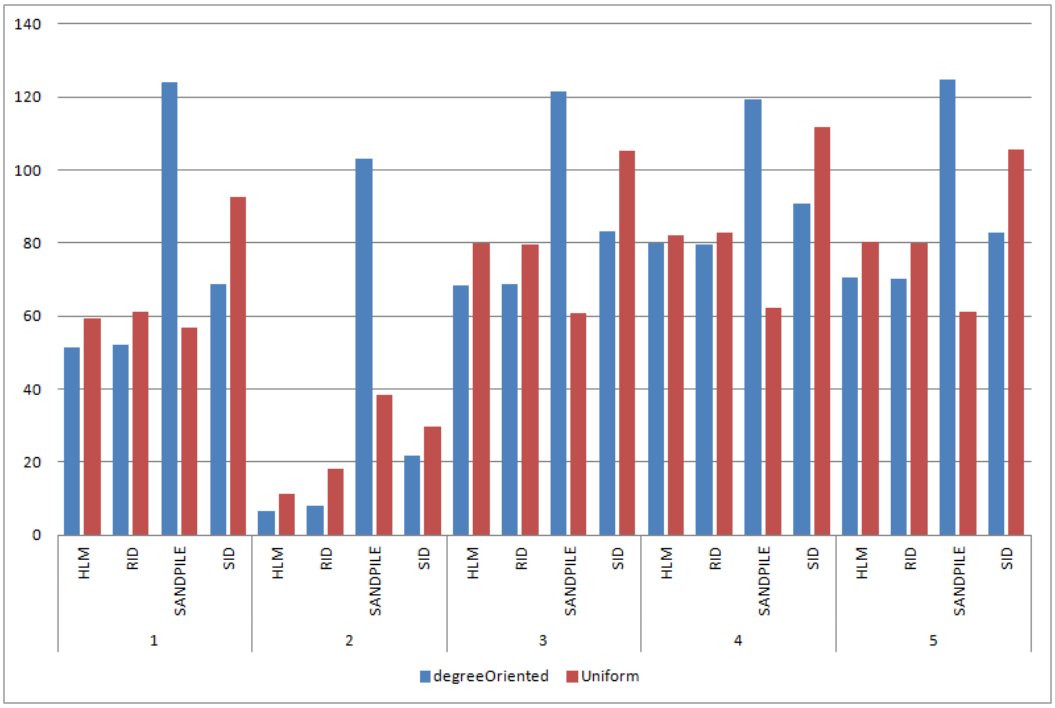


Figure 5.16: MRT using two types of distribution of arrival rate in real graphs

In all real graphs, the performance of all strategies is enhanced when degree-oriented distribution of arrival rate is used except for Sandpile balancing strategy.

For generated graphs, the performance of all load balancing strategies is enhanced in Barabàsi-Albert graph. Due to the existent of "few" hub nodes, gained enhancement in Barabàsi-Albert and real graphs is better than that in other graphs. In Random graph, positive changes for SID, no changes for HLM and Sandpile, and negative changes for RID. The system loses its stability in both Random Euclidean and Watts-Strogatz graphs for all balancing strategies.

Nodes are sorted by their degree ascending or descending according to used strategy. SID and Sandpile require descending sort i.e. node with highest degree is assigned smallest arrival rate while RID and HLM require an ascending sort. That

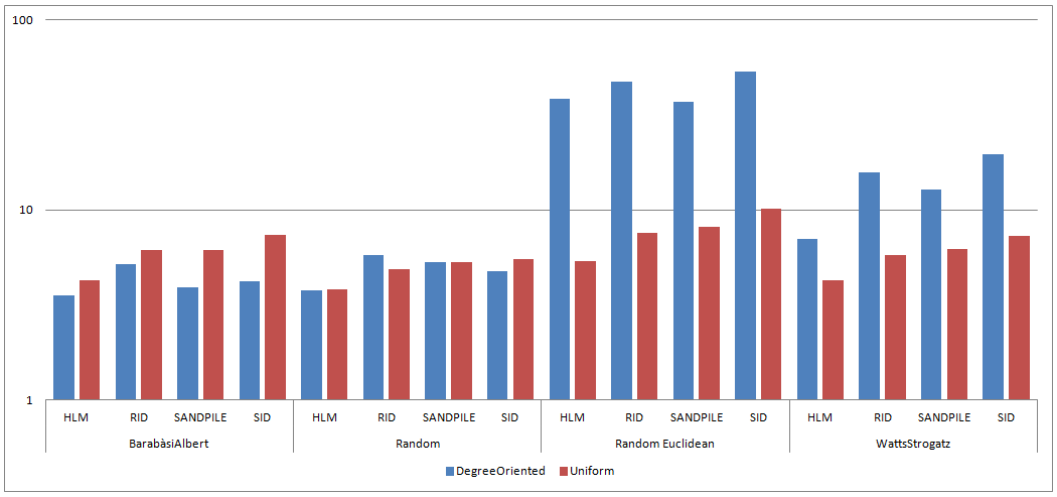


Figure 5.17: MRT using two types of distribution of arrival rate in generated graphs

is, all hub nodes become initiators. When the source (destination) is the initiator, there is no more than one destination (source) to select. If improper sorting is made, the initiator has to select from many nodes. The case in which many migrations fail because the small difference of load between the selected node and the initiator.

Finally, in Random Euclidean and Watts-Strogatz graphs, nodes of large degree composite dense clusters. Hence, nodes inside clusters have same arrival rate i.e. they are already balanced while clusters are imbalanced. Hence, degree oriented is proper for network with high degree deviation but it is improper for a high clustered networks.

In summary, the performance of the load balancing is highly affected by the network structure when it is static. The network diameter, degree deviation and average clustering coefficient playing an important role in accelerating or preventing system from reach a steady state.

5.2.3 Load balancing with global scheme resource discovery

The objective is to investigate the impact of dynamic overlay network on the system behavior. Same configuration of local scheme is used except that information is delivered using global scheme. A node may have information from direct and non-direct neighbor ones. Two methods are tested for this scheme: rumor spreading or mobile agent based resource discovery methods.

Load balancing in these tests depends on the structure of the overlay network. Overlay network is dynamic since the content of local cache is subject to change each cycle.

The accuracy and amount of items in the cache depends on TTL value and chosen cache size. Tests in this section use $TTL = 3$. For cache size six values are chosen for test: 4, 8, 16, 24, 32, and 48. That means the average out-degrees of the overlay network will be like the average degrees of the underlay networks.

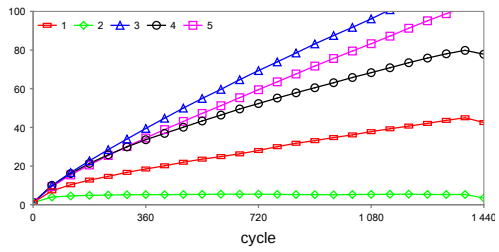
Each underlay network that is represented by graph instances in table 5.2 is used by resource discovery methods. Each time, rumor spreading or mobile agent based discovery methods is used. The characteristics of overlay network structure are investigated in section 5.2.5.

5.2.3.1 Real graphs

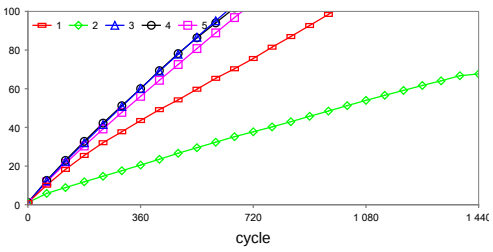
First, real graphs are considered as underlay networks. Figures 5.18, 5.19, 5.20 and 5.21 show obtained σQ. The first pair of figures is for cache size equals to 4 while the second is for cache size equals to 32. Rumor spreading and mobile agent based discovery methods are respectively used in each pair of figures. Each sub-figure shows a load balancing strategy.

Small cache size makes it difficult for all graphs (except graph 2) to reach an equilibrium point. Large cache size, e.g. 32, let Sandpile load balancing strategy reach a steady state in all graphs.

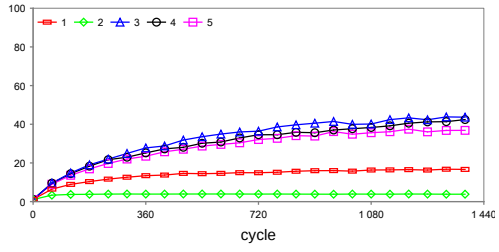
Rumor spreading gets better performance for SID and RID than for mobile agent, while mobile agents gets better performance for SANDPILE and HLM than for rumor spreading. In fact, rumor spreading based method provides recent information and guaranties that a node gets information from at least one neighbor at each cycle. Conversely, a node may not be visited by any mobile agent during one or more cycles.



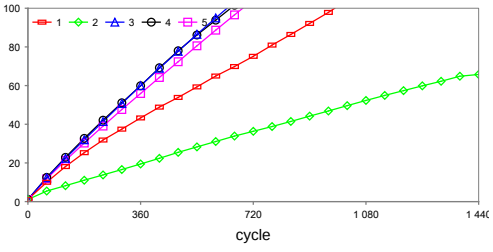
(a) SID



(b) RID

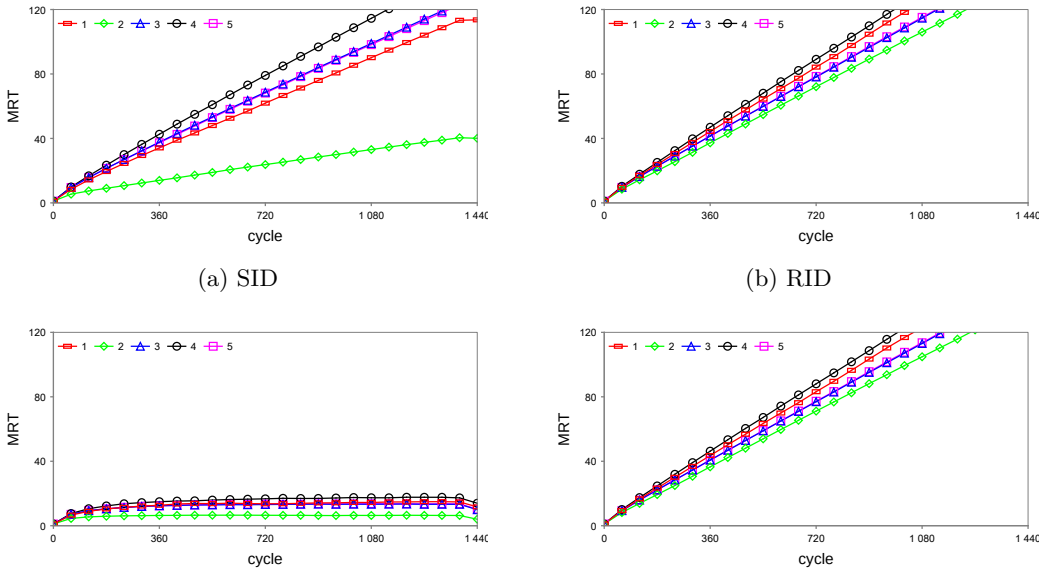


(c) SandPile



(d) HLM

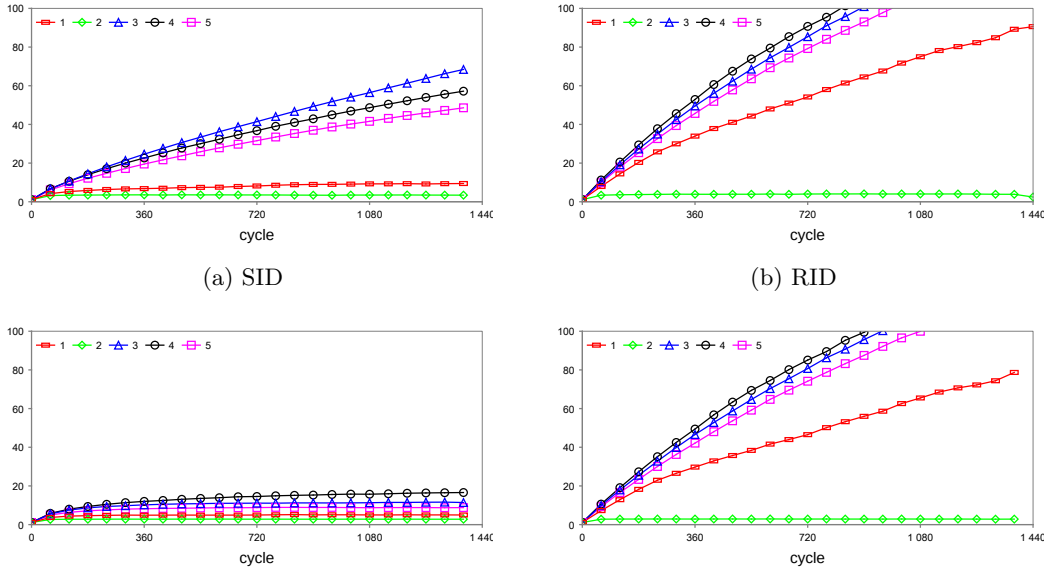
Figure 5.18: MRT in real graphs using rumor spreading, cache size=4, and Identical job type



(a) SID (b) RID

(c) SandPile (d) HLM

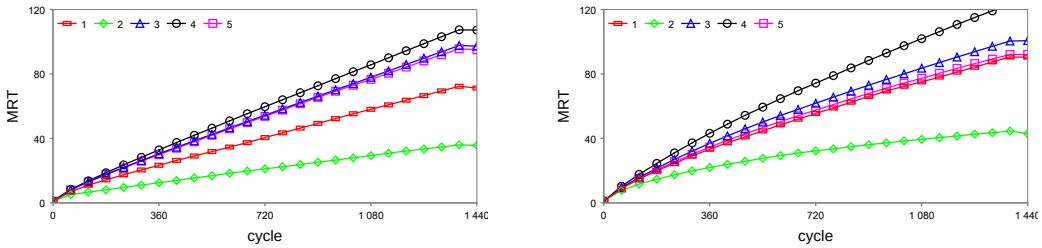
Figure 5.19: MRT in real graphs using mobile agent, cache size=4, and Identical job type



(a) SID (b) RID

(c) SandPile (d) HLM

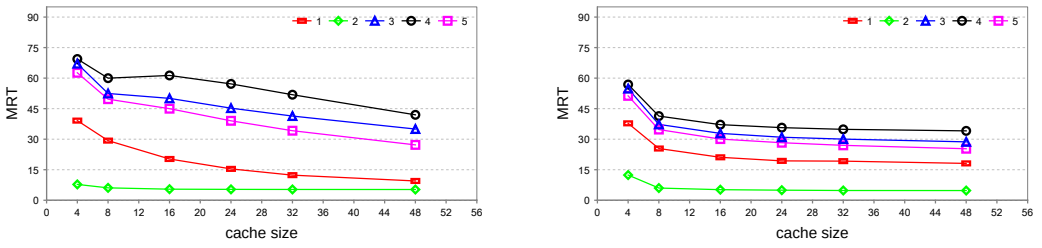
Figure 5.20: MRT in real graphs using rumor spreading, cache size=32, and Identical job type



(a) SID (b) RID

(c) SandPile (d) HLM

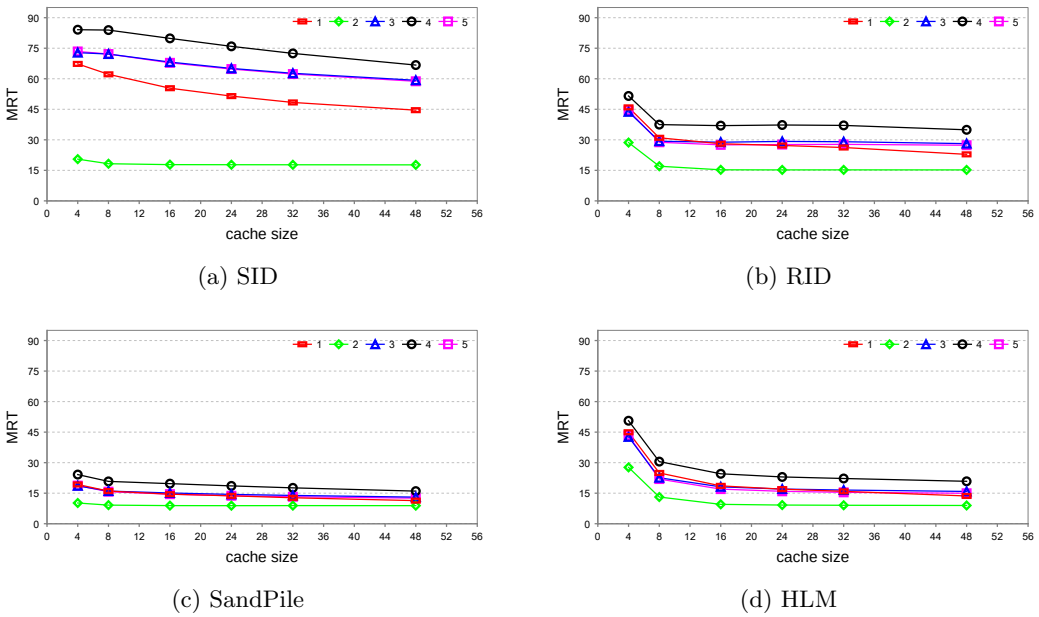
Figure 5.21: MRT in real graphs using mobile agent, cache size=32, and Identical job type



(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.22: Total MRT in real graphs using rumor spreading for using Identical job type



(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.23: Total MRT in real graphs using mobile agent for using Identical job type

Figure 5.22 and 5.23 summarize total MRT obtained in real graphs for each load balancing strategy using rumor spreading and mobile agents based resource discovery methods respectively. Result is computed for all cache sizes.

In all graphs and all load balancing strategies, obtained MRT is better for using rumor spreading than using mobile agent based method.

SID, Sandpile, and RID enhance their performance when cache size is increased. HLM gets no improvement for increasing cache size more than 16.

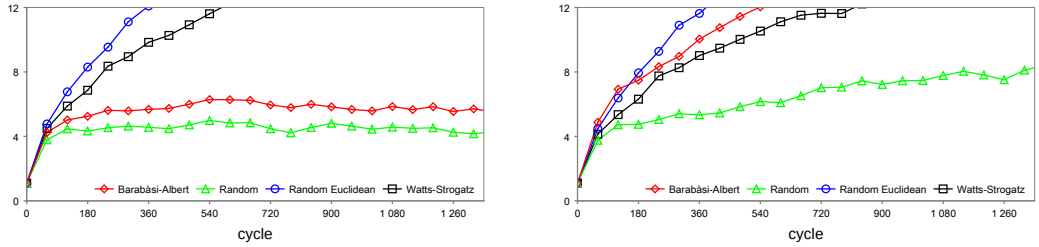
A poor performance is noticed in a three load balancing strategies for four graphs. Sandpile is the best load balancing strategy. The structure of the created overlay network does not convent for SID, RID, and HLM because it has large number of components, which is a result of large clustering coefficient and large degree deviation for all graphs except 2. The characteristics of overlay network created on real graphs are presented in 5.2.5.

5.2.3.2 Generated graphs

Now, generated graphs are tested. We first investigate underlay graphs of small average degree. Figures 5.24 and 5.25 show σQ, figures 5.26 and 5.27 show the number of waiting jobs. The result are for cache size equals 4. Figures 5.28 and 5.29 show σQ for graph of average degree 8.

Results show that when a small cache size, e.g. 4, is used an equilibrium is not reachable for both discovery methods. Only Sandpile load balancing strategy could reaches a steady state in both discovery methods. SID, RID, HLM send or receive one job for each migration. While for Sandpile, source and destination nodes may

send or receive many jobs respectively. Moreover, even if a highly overloaded (or underloaded) is not concerned by migration during several cycles, its load may be deeply changed during one cycle.



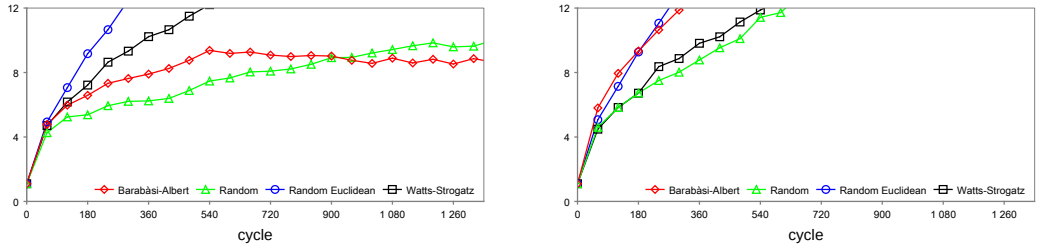
(a) SID

(b) RID

(c) SandPile

(d) HLM

Figure 5.24: σQ for of using rumor spreading in generated graphs of $\overline{deg}(G) = 4$, cache size =4, and Identical job type.



(a) SID

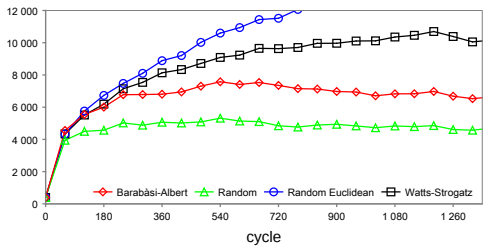
(b) RID

(c) SandPile

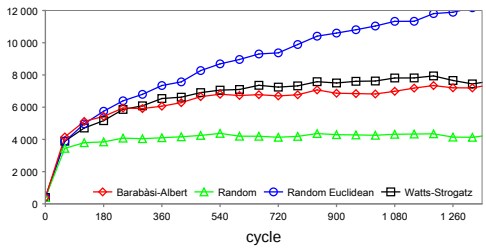
(d) HLM

Figure 5.25: σQ for using mobile agents in generated graphs of $\overline{deg}(G) = 4$, cache size=4, and Identical job type.

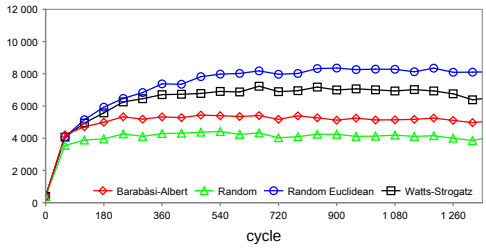
Increasing cache size a little bit, e.g. to 8, changes the states of the system in both discovery methods. Except for Random Euclidean graphs and SID load balancing, the system converges to a steady state in all other models.



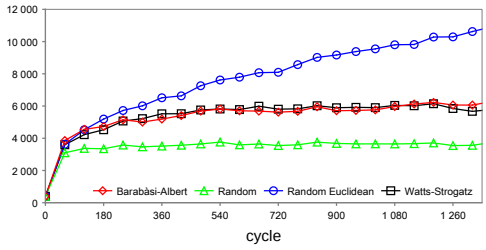
(a) SID



(b) RID

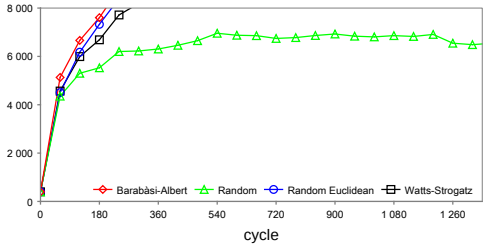


(c) SandPile

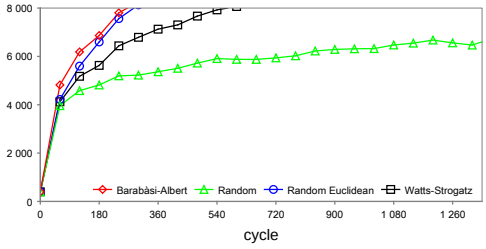


(d) HLM

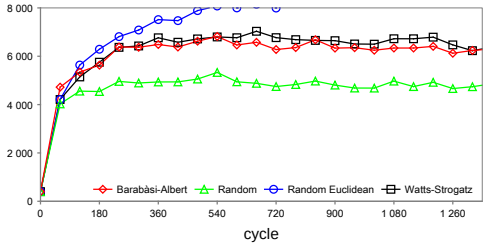
Figure 5.26: waiting jobs using rumor spreading in generated graphs of $\overline{deg}(G) = 4$, cache size =4



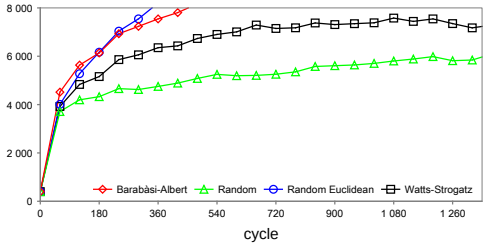
(a) SID



(b) RID



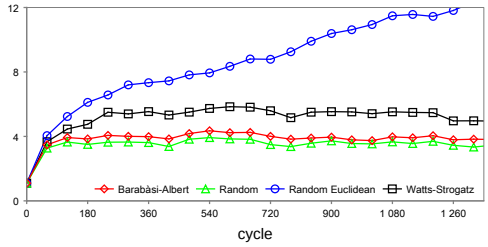
(c) SandPile



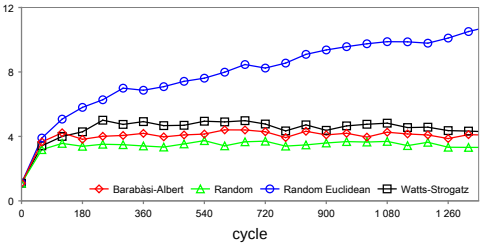
(d) HLM

Figure 5.27: waiting jobs using mobile agents in generated graphs of $\overline{deg}(G) = 4$, cache size =4

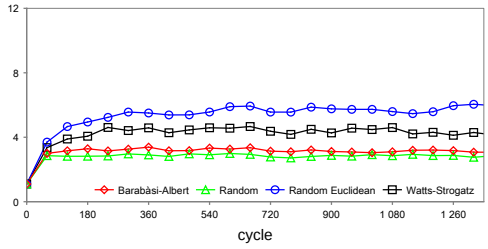
Using cache size 16 makes the system reach a steady state whatever the resource discovery method or the load balancing strategy see figures 5.30 and 5.31. However, methods vary in their performances.



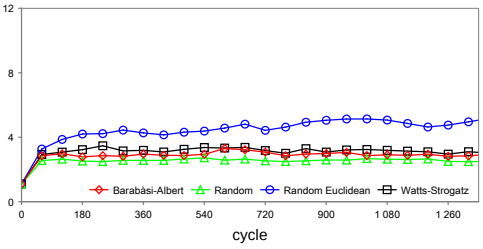
(a) SID



(b) RID

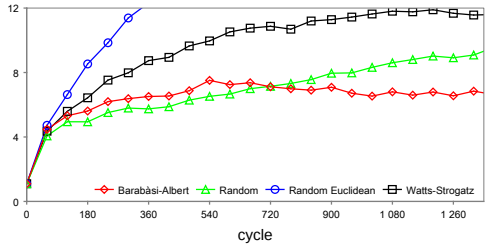


(c) SandPile

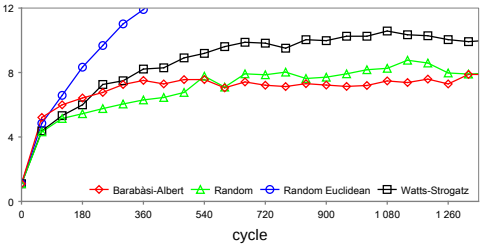


(d) HLM

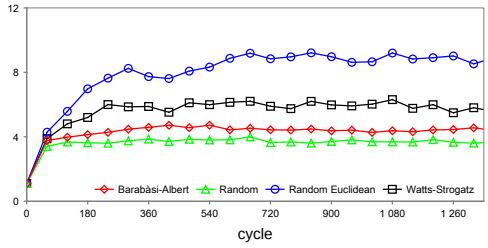
Figure 5.28: σQ in generated graphs of $\overline{deg}(G) = 4$ for of using: rumor spreading, cache size =8, and Identical job mode.



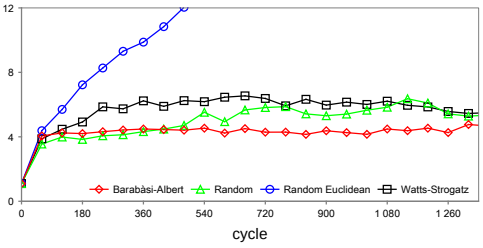
(a) SID



(b) RID

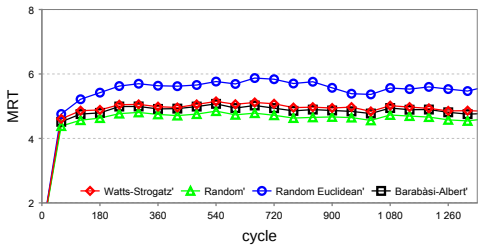


(c) SandPile

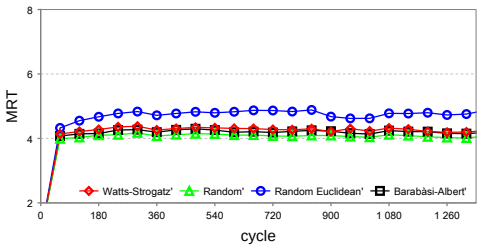


(d) HLM

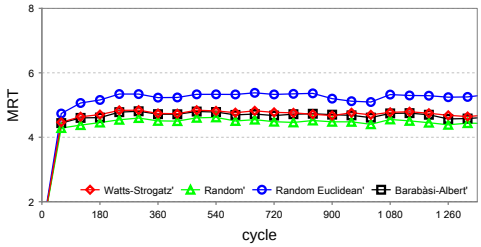
Figure 5.29: σQ in generated graphs of $\overline{deg}(G) = 4$ for using: mobile agents, cache size=8, and Identical job type.



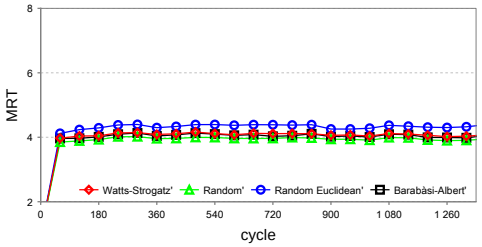
(a) SID



(b) RID

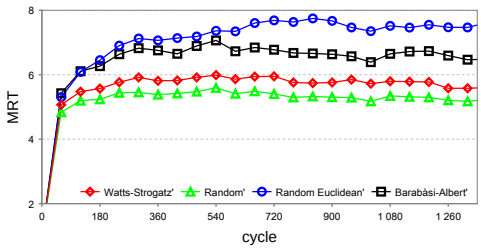


(c) SandPile

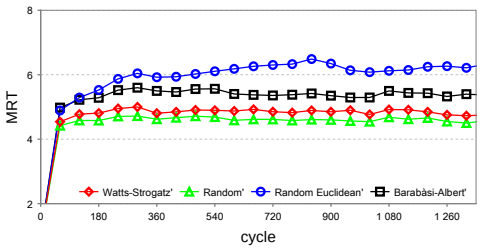


(d) HLM

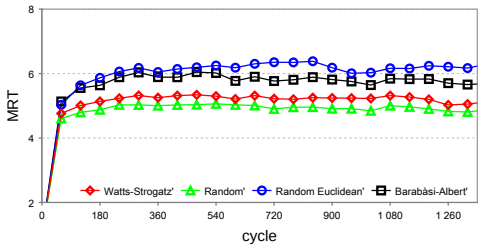
Figure 5.30: Totoal MRT in generated graphs using rumor spreading with cache size=16



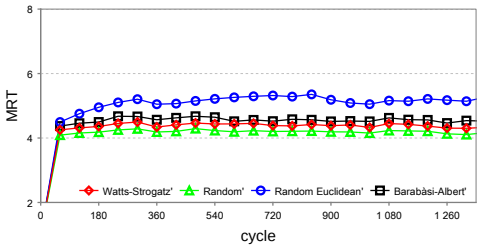
(a) SID



(b) RID



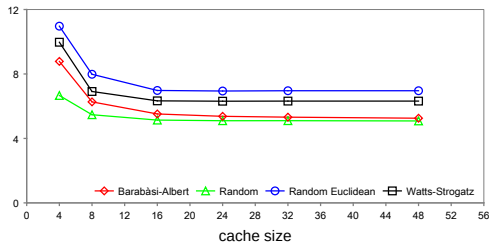
(c) SandPile



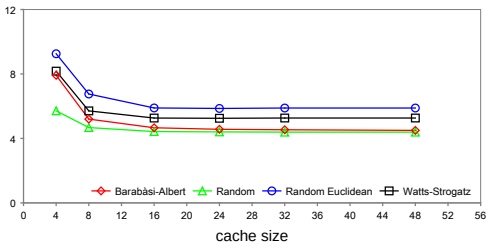
(d) HLM

Figure 5.31: Total MRT in generated graphs using mobile agent and cache size=16

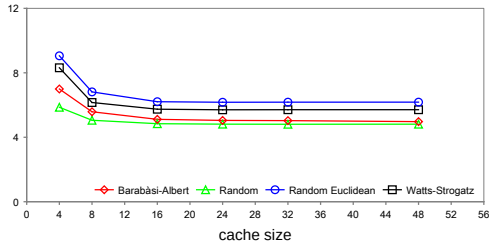
Figures 5.32 and 5.33 show Total MRT for the four load balancing strategies in generated graphs of average degree 4. Result shown total MRT computed for five cache sizes. Tests use rumor spreading and mobile agent resource discovery methods.



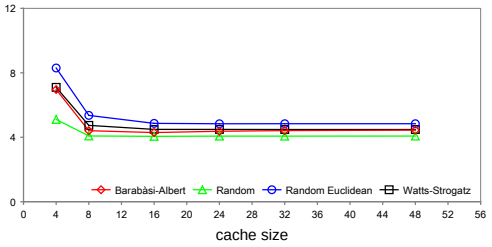
(a) SID



(b) RID

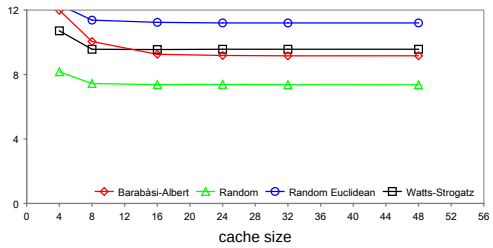


(c) SandPile

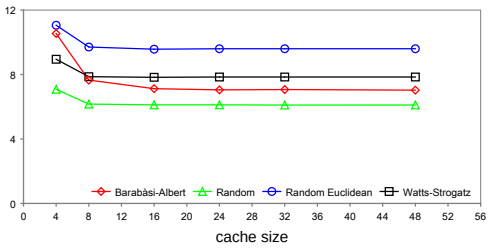


(d) HLM

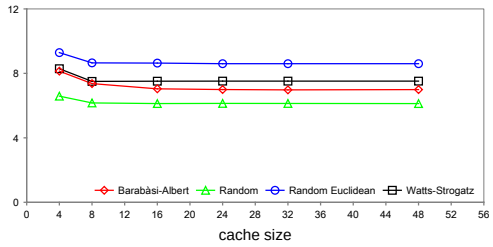
Figure 5.32: Total MRT in generated graphs of $\overline{deg}(G) = 4$ using rumor spreading with each cache size



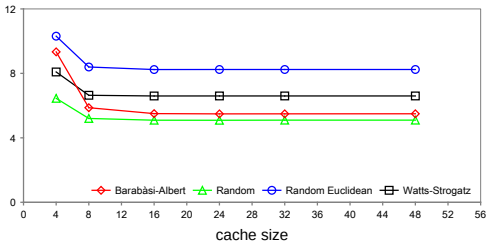
(a) SID



(b) RID



(c) SandPile



(d) HLM

Figure 5.33: Total MRT generated graphs of $\overline{deg}(G) = 4$ using mobile agent with each cache size

In global scheme, the view of a node becomes larger than its direct neighbors. The resulting overlay network structure is dynamic since the cache content is dynamic in both size and the belonging items. Cache size 16 is more suitable for all

strategies and underlay networks. Indeed, figures show increasing cache size does not enhances (or just a little bit) the total MRT.

5.2.4 The impact of resource discovery method

Figures 5.34, 5.35, 5.36, and 5.37 show the impact of each resource discovery method on the performance of SID, RID, Sandpile, and HLM load balancing strategy respectively. Shown results are for all average degrees. The used cache size is the same of the average degree of underlay graph.

A resource discovery method increases or decreases system performance according to two factors: underlay graph model and its average degree (or cache size). Local scheme is the best resource discovery method on Random graph for all average degree and load balancing strategies.

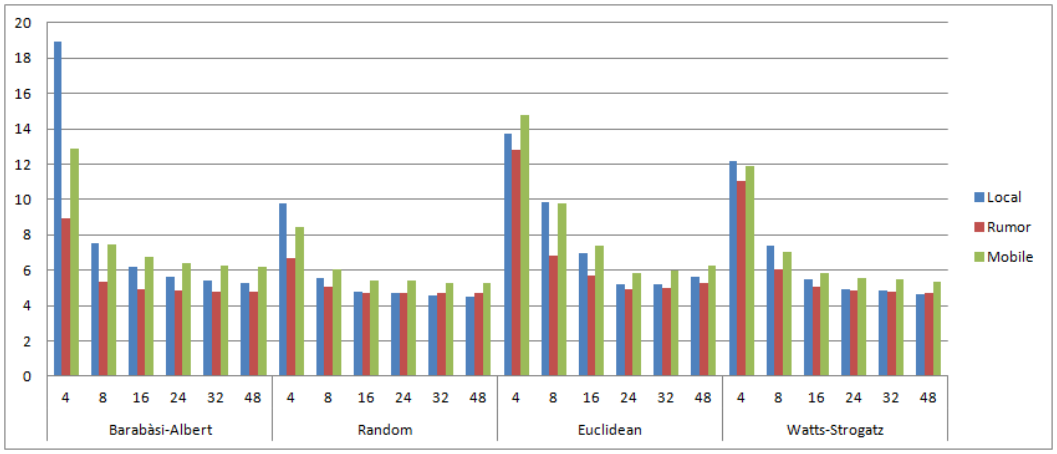


Figure 5.34: Total MRT on generated graphs using SID with each resource discovery method

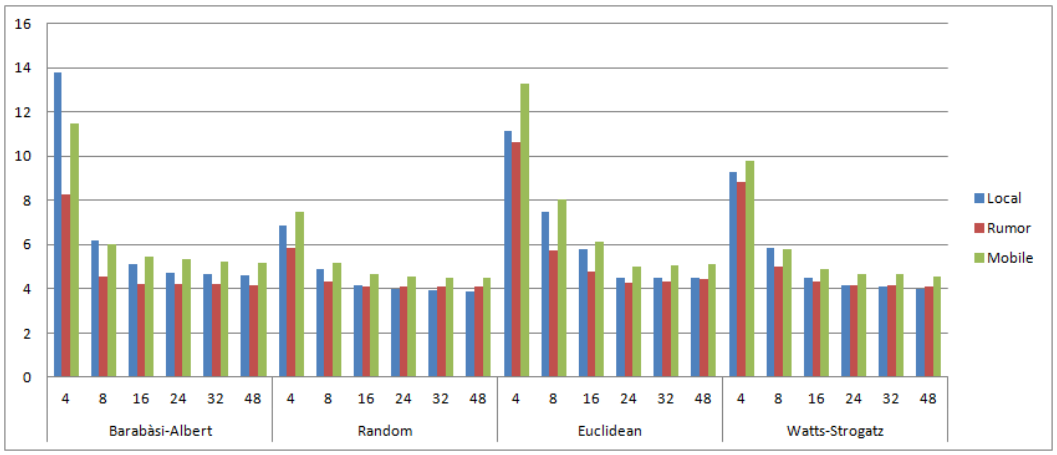


Figure 5.35: Total MRT on generated graphs using RID with each resource discovery method

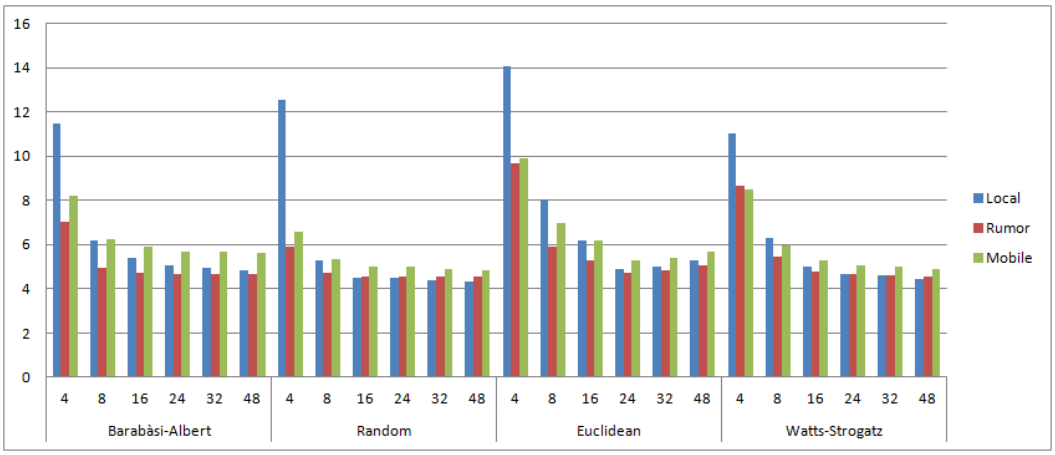


Figure 5.36: Total MRT on generated graph using SANDPILE with each resource discovery method

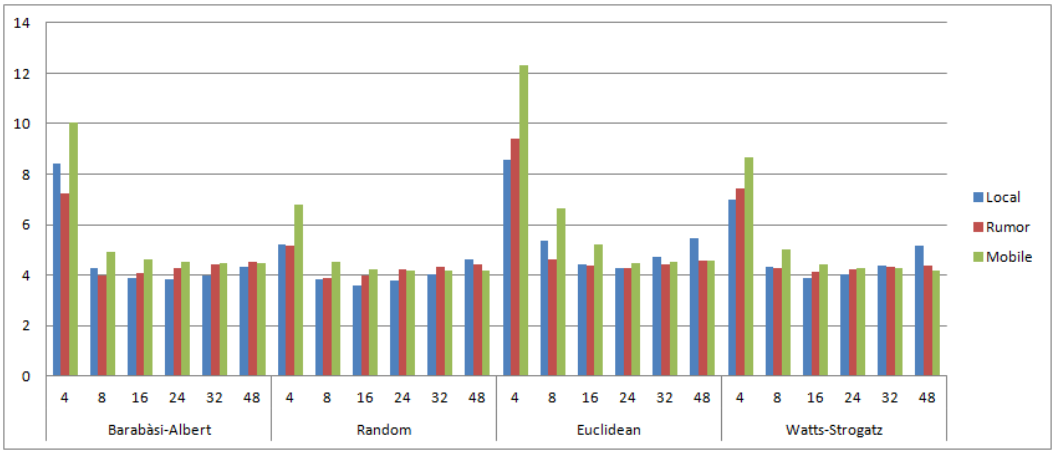


Figure 5.37: Total MRT on generated graph using HLM with each resource discovery method

Best Total MRT is obtained using rumor spreading based resource discovery method in most other cases. Then local scheme and mobile agent are come respectively. SID, RID, and Sandpile behave similarly for each resource discovery method. For HLM load balancing strategy, rumor spreading is the best for small average degree then local scheme is the best in high average degree. As explained in previous sections, HLM prefers a small average degree (cache size).

5.2.5 Structure of the overlay network

The structure of overlay network may depend on many parameters like the underlay network structure, discovery method, information aging (TTL), and K_D local cache size.

Rumor spreading							
Graph	WCC	$d(G)$	$\Delta(G)$	$\delta(G)$	$\sigma(G)$	$\overline{deg}(G)$	$\overline{C}$
1	13	10	31	0	23	11	0.448
2	8	8	31	1	18	9	0.362
3	44	15	31	1	26	9	0.509
4	11	11	31	1	27	9	0.516
5	61	15	31	1	27	9	0.491
mobile agents							
Graph	WCC	$d(G)$	$\Delta(G)$	$\delta(G)$	$\sigma(G)$	$\overline{deg}(G)$	$\overline{C}$
1	465	10	31	0	7	11	0.206
2	1430	11	31	0	6	7	0.183
3	3398	19	31	0	8	12	0.196
4	1899	16	31	0	9	12	0.171
5	5698	19	31	0	8	12	0.194

Table 5.5: Characteristics of overlay network created by each discovery method over real graphs(out-degree is considered)

Local scheme keeps same structure of underlay network (so no need to consider here). Mobile agent and rumor spreading based discovery methods differ in their basics. They create different structures of the overlay network for same environment setting. All above instances of graphs are tested as underlay networks instances. Two important parameters have to be determined in each test: TTL and K_D.

Six cache sizes (4, 8, 16, 24, 32, 48) and nine TTL values (2-10) are tested. $TTL = 1$ is not considered because information expires before spreading to new neighbors. Since the overlay is dynamic in such discovery methods, a snapshot is used for the analysis. A snapshot is taken when the degree distribution becomes stable (often, after less than 100 cycles). However, all snapshots are taken after 720 cycles.

In addition to degree distribution, other characteristics like diameter, average clustering coefficient, deviation of degree, etc. are also computed.

5.2.5.1 Real graphs

The created overlay network over real networks have a power-law degree distribution. So it is better to use Inverse Cumulative Degree Distribution $ICDD(k)$ drawing method. Figures 5.38 and 5.39 show the in-degree distribution of overlay network using rumor spreading and mobile agent methods respectively. Shown result are for using cache size equal to 32. Other characteristics of resulting overlay networks are shown in table 5.5. Both discovery methods create overlay network with many weakly connected components WCC (with significant larger WCC for using mobile agents), large degree deviation, and large diameter in all graphs except 2. Hence, it is difficult to move load quickly between nodes.

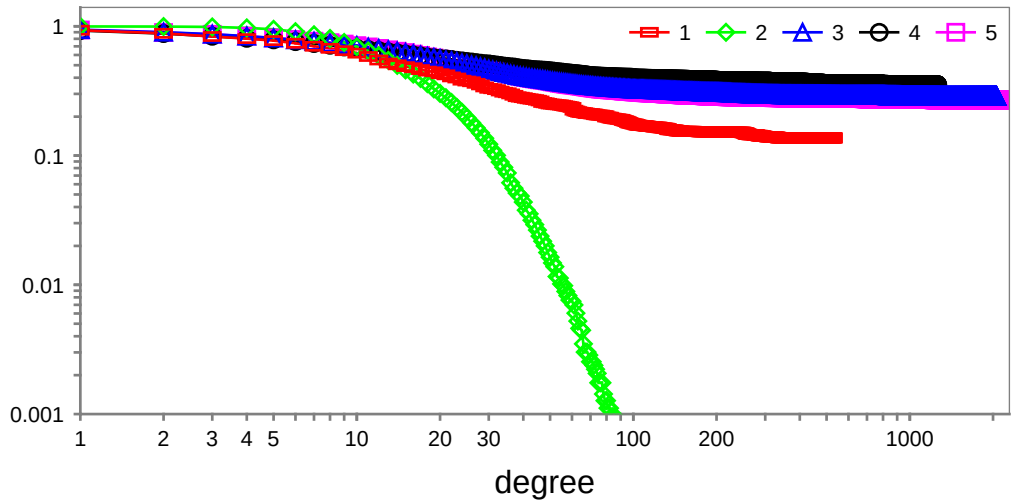


Figure 5.38: In-Degree distribution of overlay networks created on real graphs by rumor spreading method

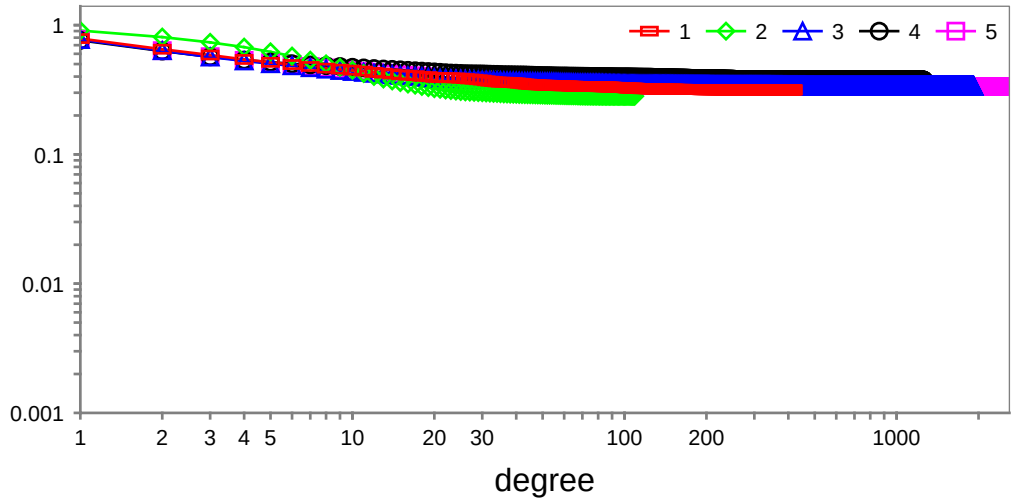


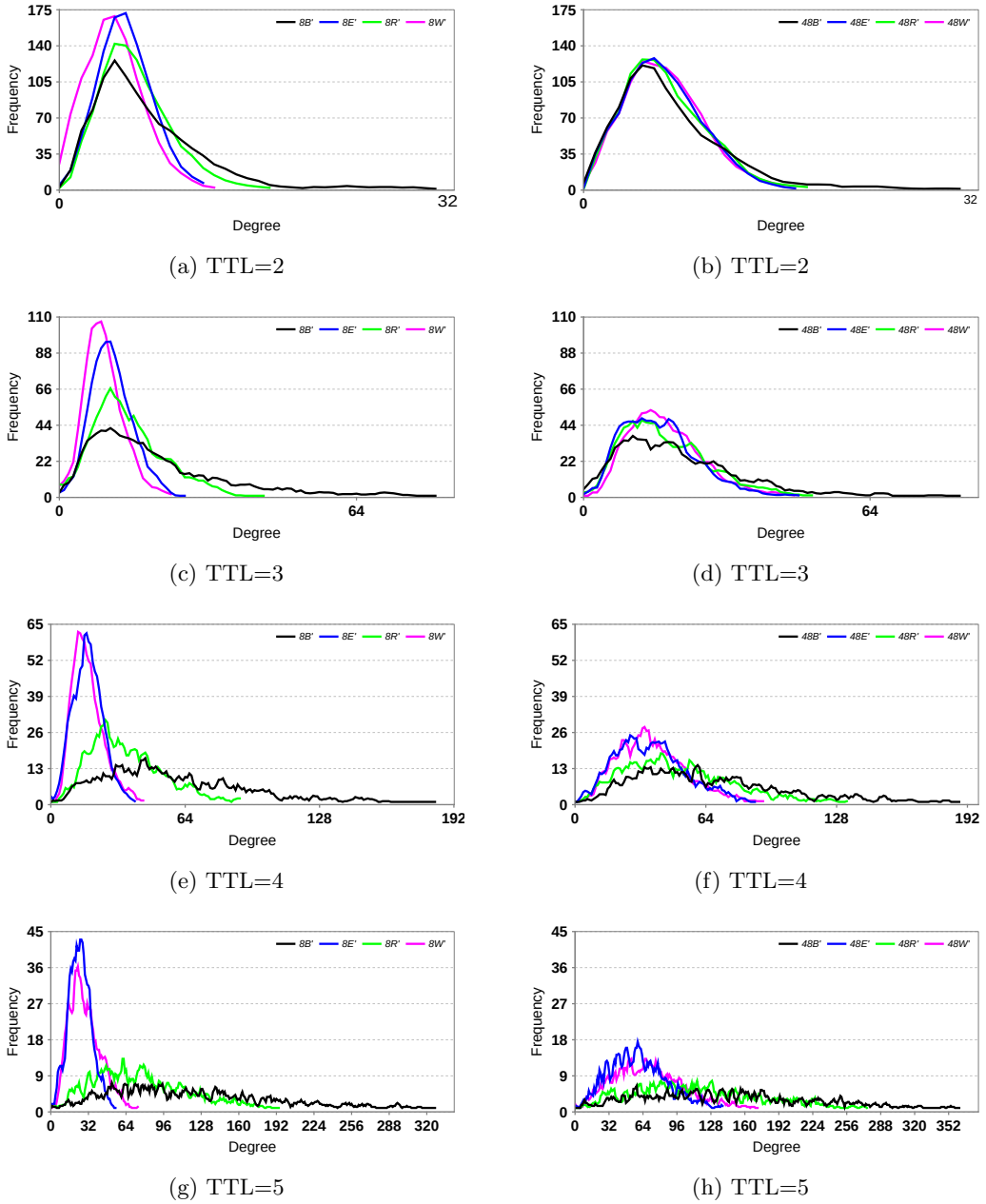
Figure 5.39: In-Degree distribution of overlay networks created on real graphs by mobile agent method

5.2.5.2 Generated graphs

Degree distribution of several snapshots of overlay network are given in figures 5.40 and 5.41. Unlimited K_D is used for result in the first figure.

Increasing TTL value stretches or shrinks the distribution curve according to the clustering coefficient of the underlay network. The number of information items that are delivered to a node is proportional to the TTL value and its degree in underlay network. Because large TTL lets information travel far and large average degree

increase the probability that a node is contacted by a neighbor for information exchange. Increasing TTL has no meaning when the K_D is small, see figure 5.41, since information is dropped even before it reaches its time limit.



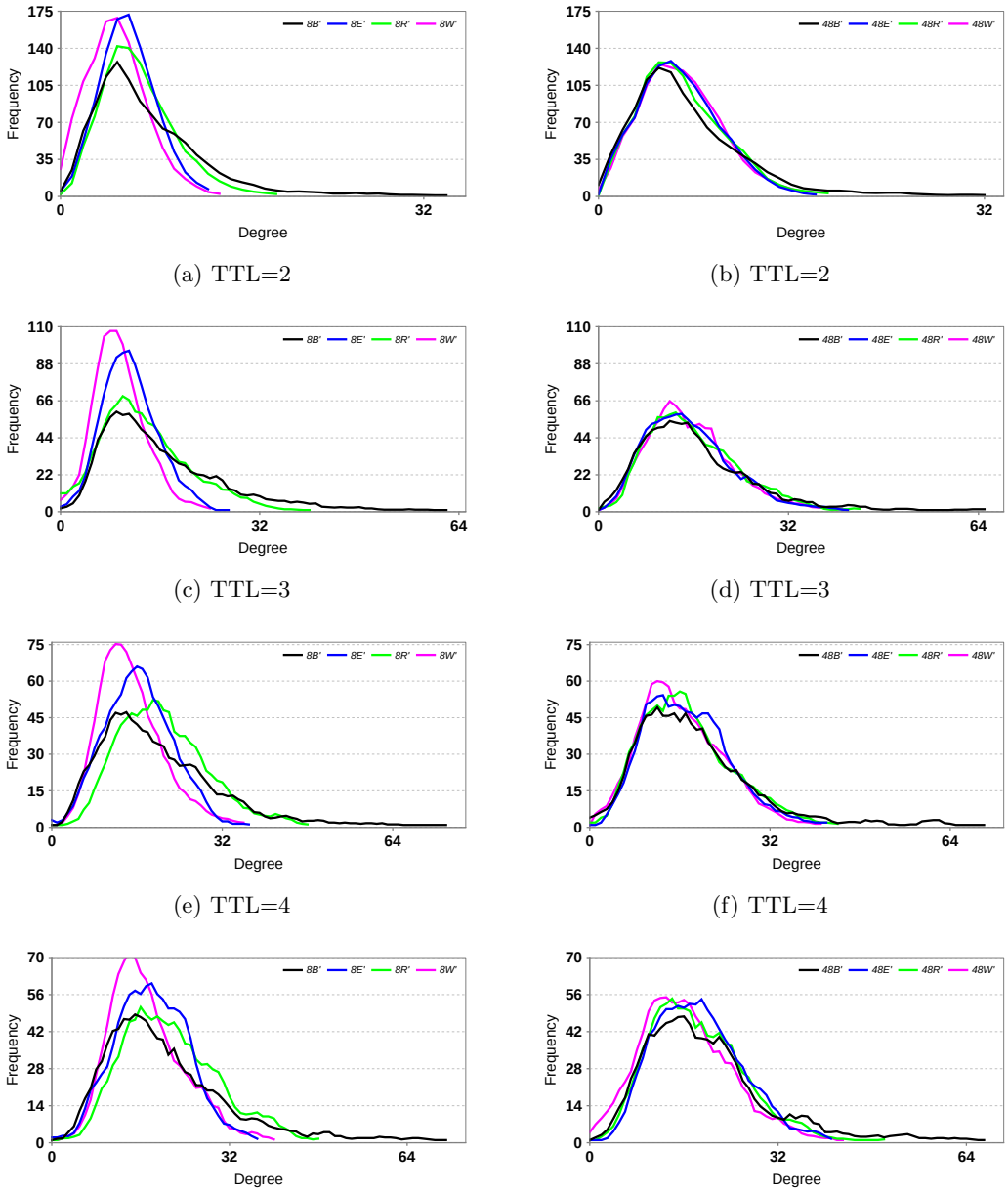
(a) TTL=2 (b) TTL=2

(c) TTL=3 (d) TTL=3

(e) TTL=4 (f) TTL=4

(g) TTL=5 (h) TTL=5

Figure 5.40: In-Degree distribution of snapshots of overlay network for $TTL = 2$ to 5: Left for underlay network of $\overline{deg}(G) = 8$, right for underlay network $\overline{deg}(G) = 48$, Rumor spreading, and $K_D = \infty$



(a) TTL=2 (b) TTL=2

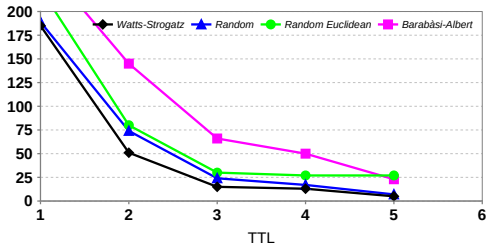
(c) TTL=3 (d) TTL=3

(e) TTL=4 (f) TTL=4

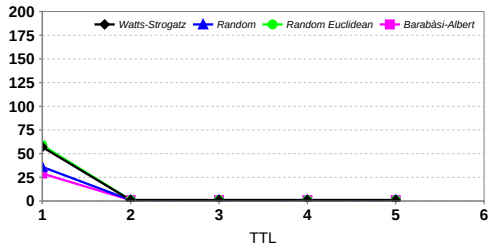
(g) TTL=5 (h) TTL=5

Figure 5.41: In-Degree distribution of overlay networks:Left for underlay network of $\overline{deg}(G) = 8$*, right for underlay network* $\overline{deg}(G) = 48$*, Rumor spreading, cache=24*

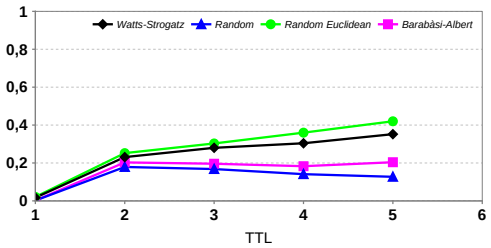
Figure 5.42 shows the number of components, average clustering coefficient $\overline{C}$ and average degree of obtained overlay networks from both methods. The underlay networks are the generated graphs of $\overline{deg}(G) = 16$. The characteristics of an overlay network resulting from rumor spreading or mobile agent resource discovery methods depend on the TTL value and cache size. Lower TTL values always give non-connected graphs.



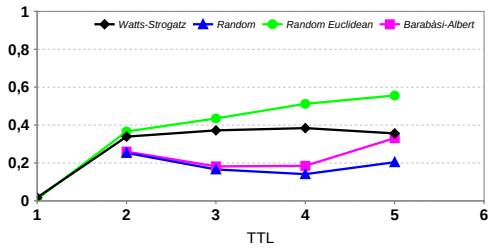
(a) number of components



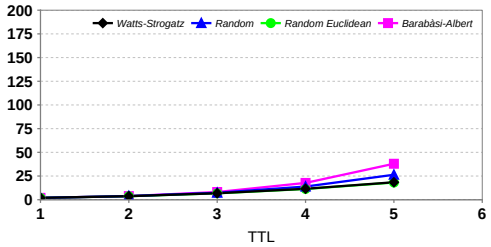
(b) number of components



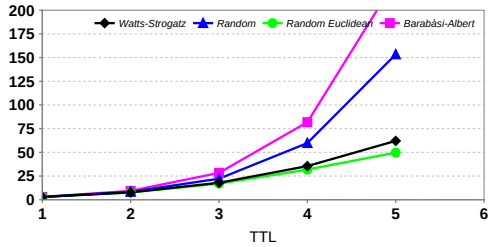
(c) $\overline{C}$



(d) $\overline{C}$



(e) Average outdegree

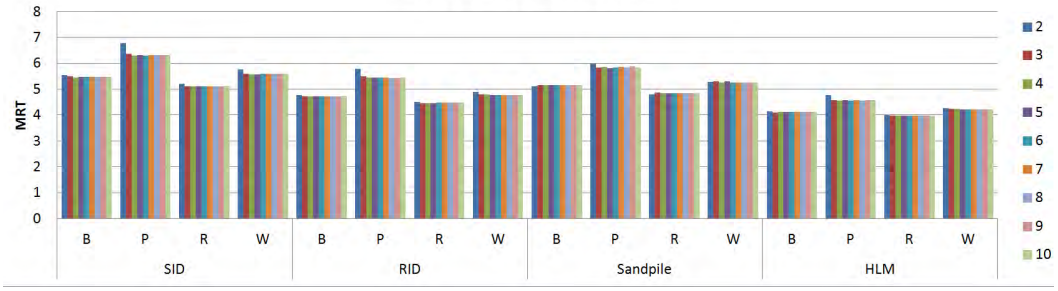


(f) Average outdegree

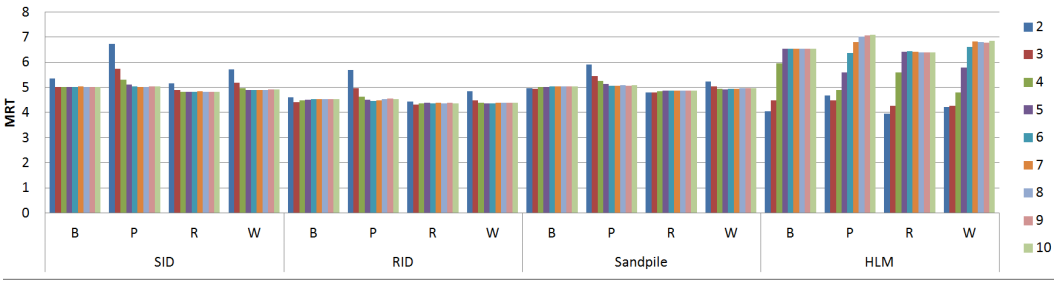
Figure 5.42: Features of overlay network on generated graphs of $\overline{deg}(G) = 16$: left from mobile agent, right from rumor spreading.

Mobile agent based discovery method differs from rumor spreading: sometimes, nodes may not be visited by mobile agent while in rumor spreading based method, a node is always concerned with one exchange at least. Hence, the overlay network may remain disconnected for values of $TTL < 5$, see figure 5.42a.

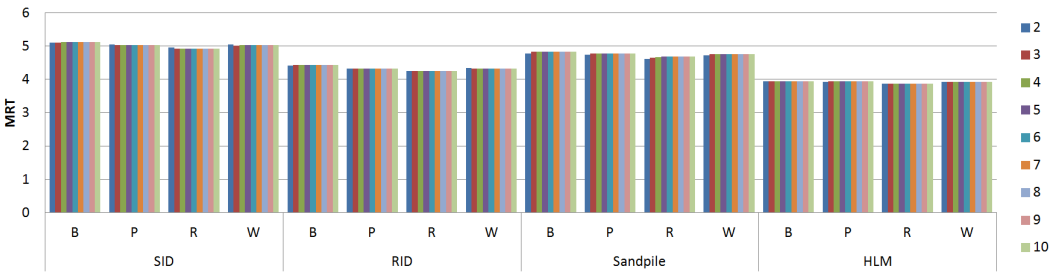
Dynamics structure of overlay network resolves the problem of non-connectivity, hence, nodes still discover other since the underlay network is connected all the time. The most important for load balancing is that each node always has a set of neighbors to cooperate with i.e the variance of average degree is small.



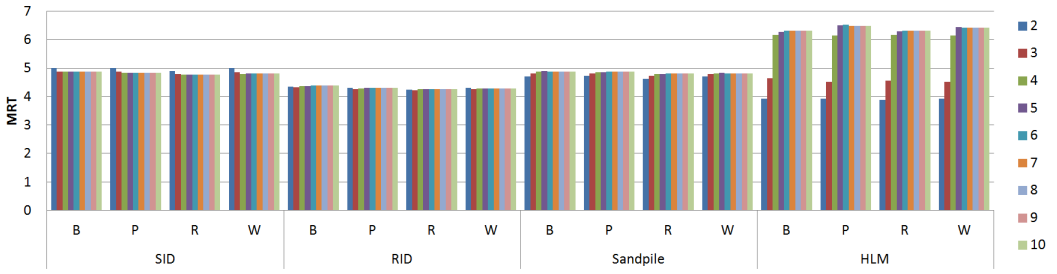
(a) degree=8, cache=8



(b) degree=8, cache=48

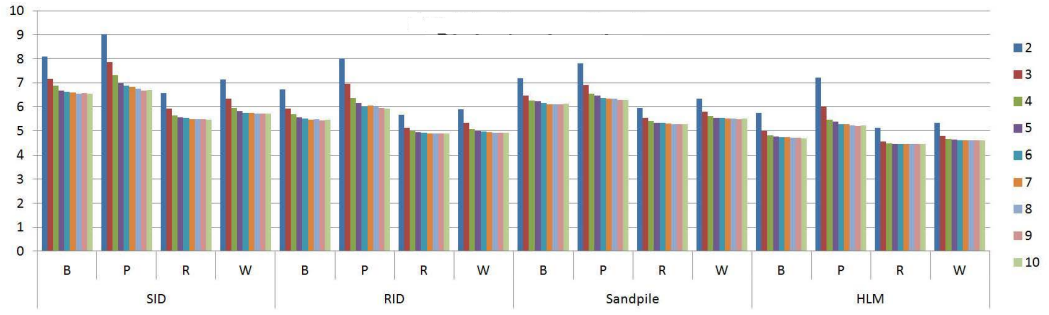


(c) degree=48, cache=8

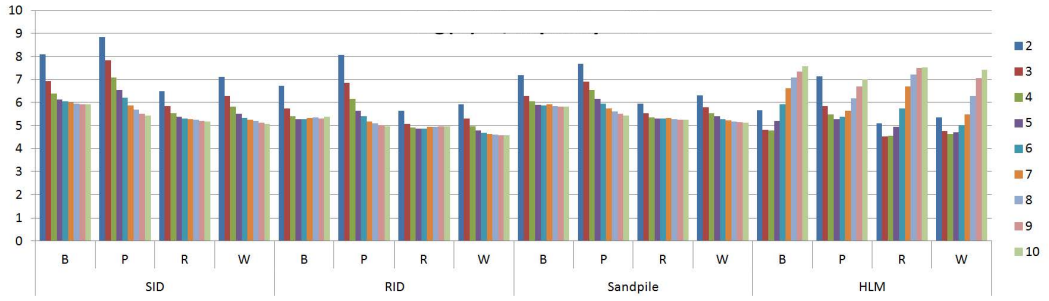


(d) degree=48, cache=48

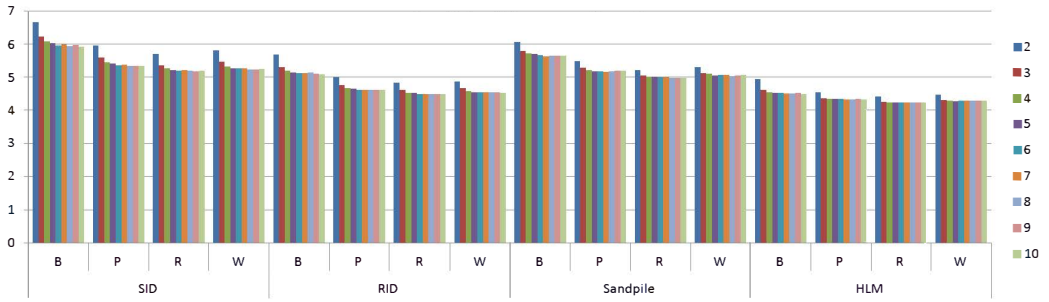
Figure 5.43: Total MRT obtained using rumor spreading by each load balancing strategy



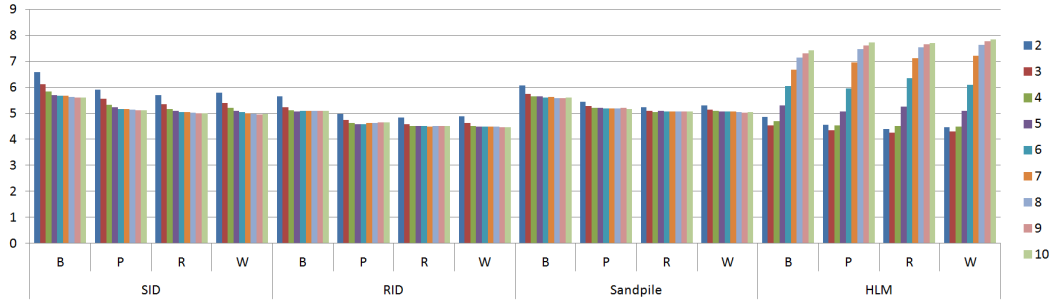
(a) degree=8, cache=8



(b) degree=8, cache=48



(c) degree=48, cache=8



(d) degree=48, cache=48

Figure 5.44: Total MRT obtained using mobile agent by each load balancing strategy

Figures 5.43 and 5.44 show MRT obtained using rumor spreading and mobile agent respectively. Shown results are for generated instance, average degrees are 8

and 48, cache sizes are 8 and 48 too. Result shows a load balancing strategy impose different cache size and TTL on each graph instance.

Cache size and TTL are each other dependent parameters. Increasing cache size has no effect when the TTL is small and vice versa. A TTL value larger than underlay graph diameter with unlimited cache size makes overlay network be a complete graph.

In the next section, it is shown that differently managing local information has a major impact, even on the overlay network structure itself.

5.2.6 Improving information management

The way by which collected information is maintained changes the structure of the overlay network and consequentially may enhance the obtained result. Two mechanisms are proposed in 3.2.2 to update cached information. A node often uses delivered information by means of resource discovery method. Information exchanged between balancing agents for validating migration process may be used to update local cache (reinforcement). Frequently selected sources or destination nodes may be preferred over other nodes when maintaining cache consistency (activeness preference).

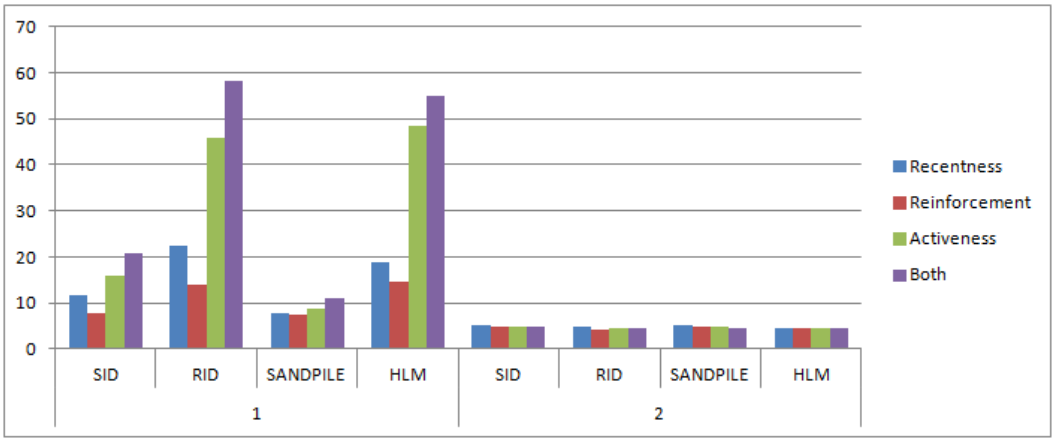


Figure 5.45: MRT resulting from different cooperation schemes in real graphs 1 and 2

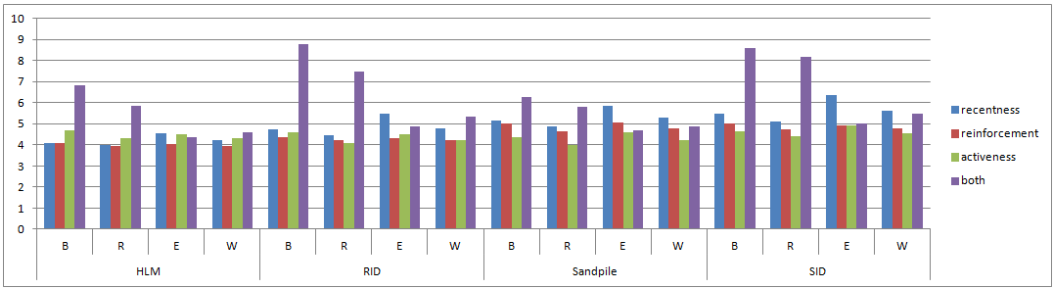
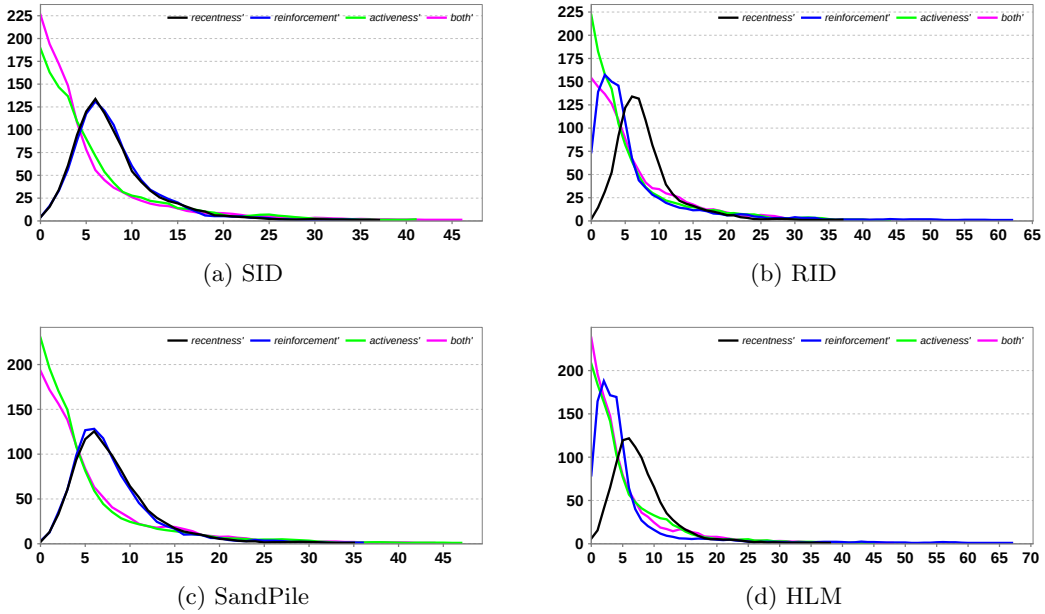


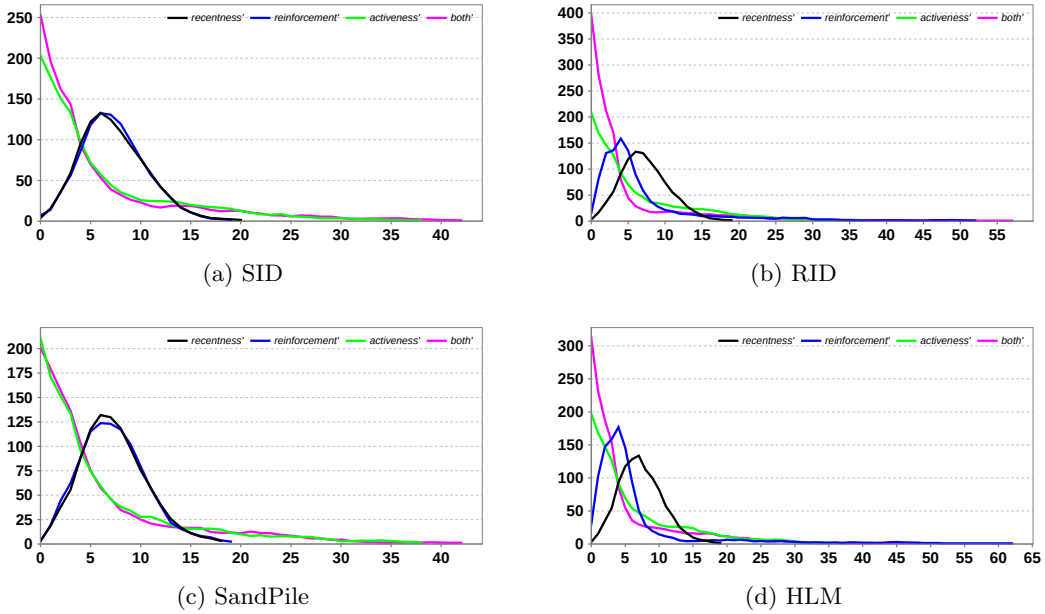
Figure 5.46: MRT resulting from different cooperation schemes in generated graphs



(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.47: In-Degree distribution of overlay network resulting from different cooperation schemes, 8B



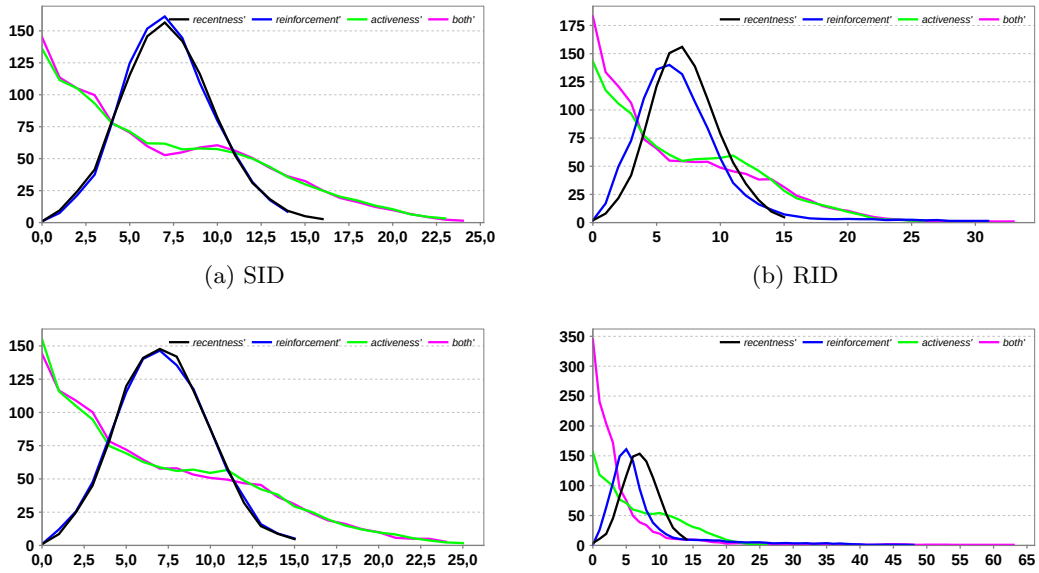
(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.48: In-Degree distribution of overlay network resulting from different cooperation schemes, 8R

Experiments of section 5.2.3 are remade using proposed mechanism of cooperation. Figure 5.45 shows obtained MRT in two real graphs 1 and 2 (graphs 3,4, and 5 give same result as 1). Shown result are for cache size equals to 32. Figure 5.46

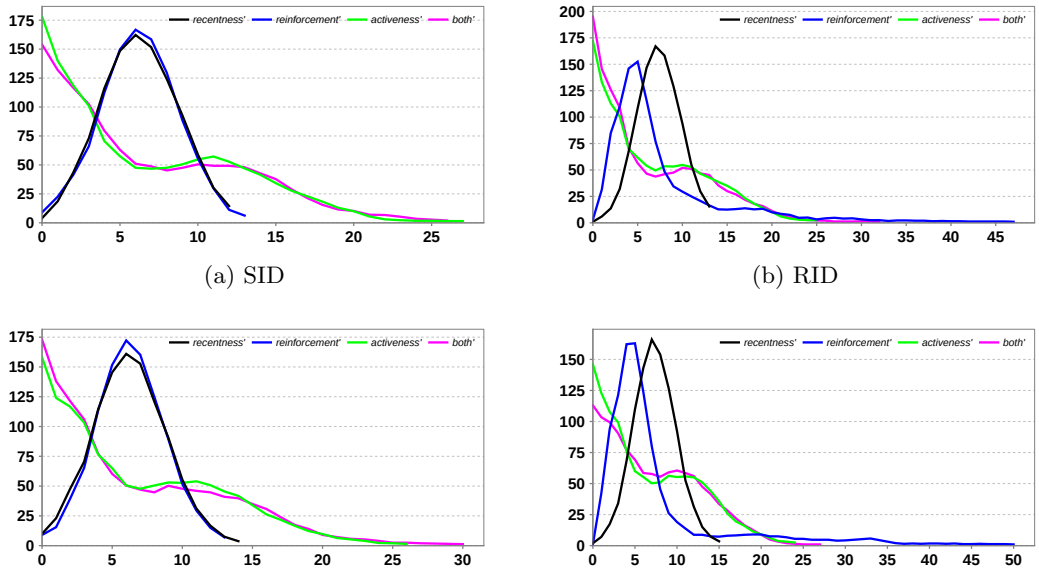
shows MRT computed using for each of cooperation schemes on different generated graphs of $\overline{deg}(G) = 8$, different strategies, Rumor spreading, and cache size equals to 8.



(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.49: In-Degree distribution of overlay network resulting from different cooperation schemes, 8E



(a) SID (b) RID

(c) SandPile (d) HLM

Figure 5.50: In-Degree distribution of overlay network resulting from different cooperation schemes, 8W

Figures 5.47, 5.48,5.49,5.50, show in-degree distribution of overlay networks on different underlay models.

It is clear that both cooperation schemes (reinforcement and activeness) enhance obtained MRT. Activeness has the most powerful impact on the total MRT in most cases. Reinforcement also makes load balancing get better performance than recentness. However, in one case, specifically with Random Euclidean graph, the Total MRT is good when both techniques are used at same time.

When information is filtered only by their age, i.e. recentness preference, a node may keep information about overloaded, underloaded or intermediately loaded nodes. In fact, overloaded (underloaded) nodes are interested by underloaded (overloaded) nodes. Hence, a node does not use two-third of cached information (on assumption that load is uniformly distributed).

The number of information and covering area are increased when non-direct neighbor nodes have up-to-date copies of it. Thus, frequently selected nodes have chances more than others to be selected again in the next load balancing decisions.

Same thing is happening for activeness preference. A node with large number of migrations is preferred over other nodes. Hence, they will be kept at the cache of nodes while their age does not exceed the TTL limit. The effect of information reinforcement vanishes after a number of cycles equals to TTL if the node is not selected anymore. While the information about active node (one with preferred activeness value) is kept until it reaches the TTL limit. However, the first technique, e.g. reinforcement spreads information of frequently selected node far to a distance from the origin node larger than the TTL value.

Recentness scheme creates a small world structure (the diameter of the overlay network is around the diameter of the underlay network and the average clustering coefficient is about 2.0). Reinforcement stretches the tail of the curve while conserving its shape. When activeness is used (alone or with reinforcement) the distribution becomes like a power-law curve. Demanded nodes (load sources) during load balancing process are located in its right part while most other nodes (load destinations) have small in-degree and certainly have arcs pointed to source nodes. Source nodes are known by most other nodes along the time. Sender initiated load balancing strategies (SID, and Sandpile) benefit less from this technique since it is highly probable that most sources chose the same destination, hence, many migration dialogues fail to complete.

5.2.7 Heterogeneous nodes and bandwidth

All previous experiments consider the system as homogeneous. To study system of heterogeneous resource in its equilibrium state, the following experiments use nodes varying in their architecture or connection bandwidth.

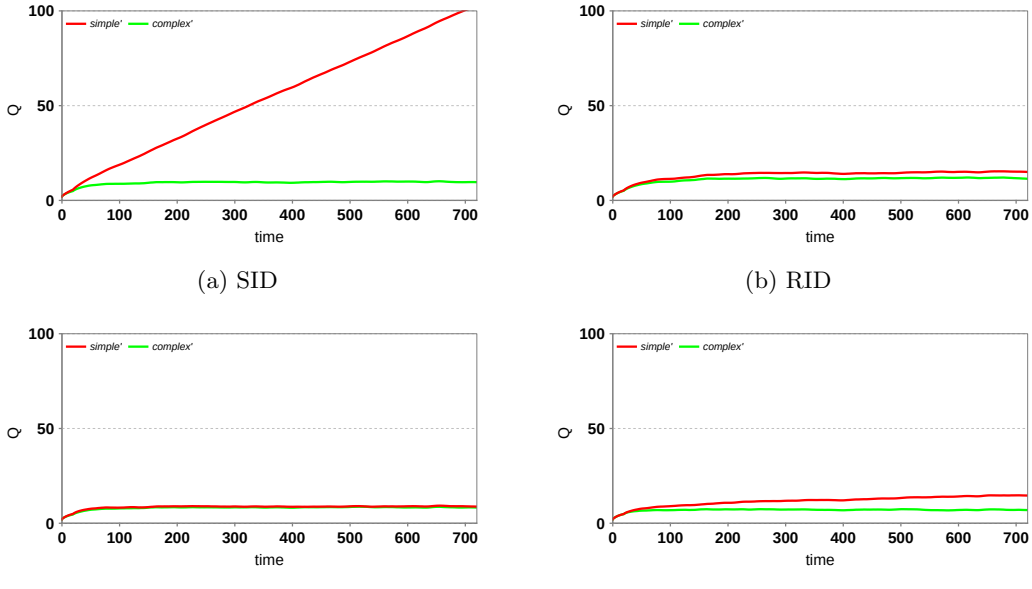
5.2.7.1 Multi community structure

Experiments are made to see impact of network structure on the system behavior in case of resources heterogeneity. The four load balancing strategies are tested. An experiment uses all underlay network structures, Rumor spreading based diffusion method. Cache size K_D is fixed to 16 items. A node in a network is randomly marked by a generic or specific community label (Node configuration A).

First, experiments are made by caching any recent information (called simple structure). No change is made for the basic overly structure. Then, same experiments are remade with caching only information of nodes that have same or generic architecture types (called complex structure). Nodes of each architecture type create a community (dense cluster) and communicate with other communities indirectly via generic types nodes that are connected to all nodes.

Figures 5.51, 5.52, 5.53, and 5.54 show σQ, number of migrated jobs, number of aborted migrations, and Total MRT obtained using the two caching techniques (simple and complex). Shown result uses same Random graph of $\overline{deg}(G) = 16$ as underlay network, cache size equals 16, and TTL value equals 3.

A steady state is reachable by all load balancing strategies using complex structure (entire knowledge is compatible for arrived jobs) while it is not reachable in three strategies using simple one (only some entries compatible for possible migrations). In all cases, complex overlay structure helps obtaining better MRT than simple structure. It does less migrations and migration subject to less cancellation (aborting).



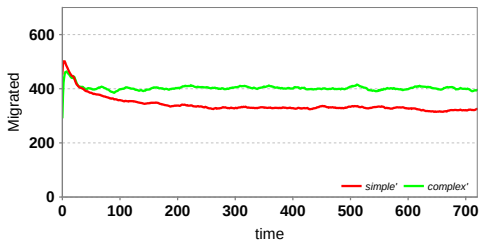
(a) SID

(b) RID

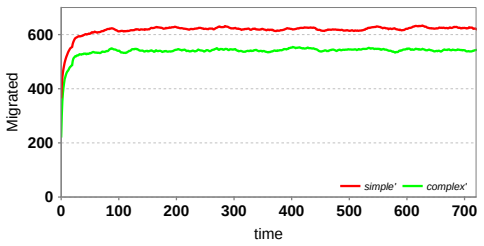
(c) SandPile

(d) HLM

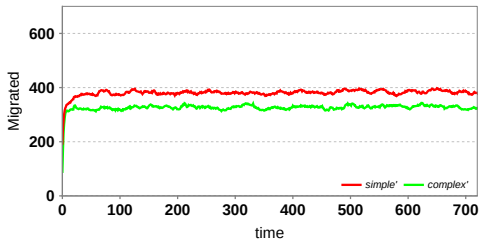
Figure 5.51: σQ of heterogeneous type using simple and complex overlay structures



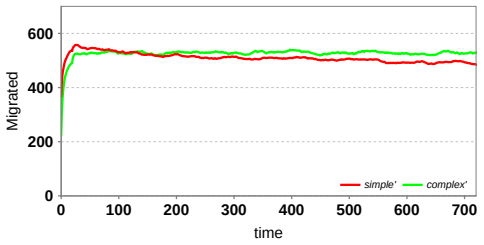
(a) SID



(b) RID

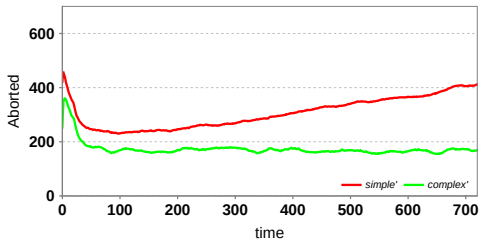


(c) SandPile

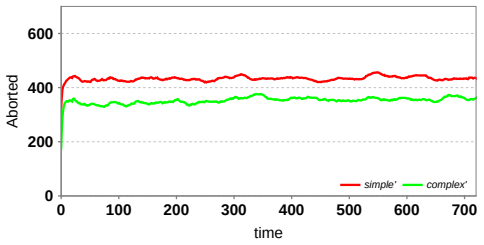


(d) HLM

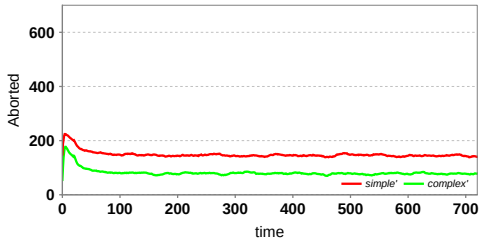
Figure 5.52: Number of migrated jobs per cycle of heterogeneous type using simple and complex overlay structures



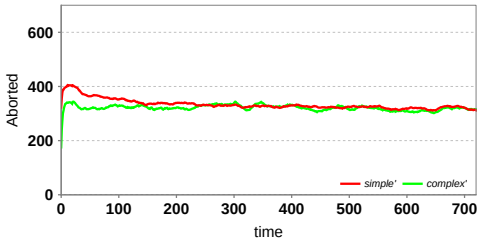
(a) SID



(b) RID



(c) SandPile



(d) HLM

Figure 5.53: Number of aborted migrations per cycle of heterogeneous type using simple and complex overlay structures

Though MRT is not enhanced much in some strategies (especially for SAND-

PILE), the same MRT is obtained using less migrations. The reason is that load balancing chooses a destination then selects a job. Hence, in simple structure a node may be selected and the migration is aborted because it is discovered as a non-compatible node. In complex overlay networks, all information in the cache are compatible nodes since the filter does it early.

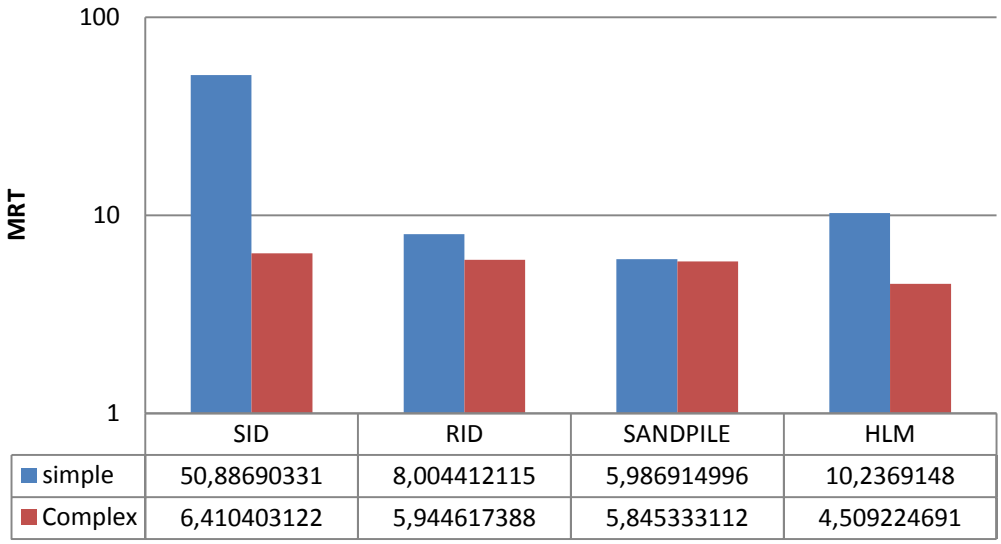


Figure 5.54: Total MRT of heterogeneous type using simple and complex overlay structures

5.2.7.2 Job size and bandwidth considerations

A job is selected for migration when the queue length of destination node plus the transmission time of the job is less than expected waiting time of that job in the source node. So far, both bandwidth and job size are considered identical for all nodes and jobs respectively. Neither the connection bandwidth nor the job size affects job selection criteria and transmission time. The time required to move any job to any node was considered one cycle.

In this section, different job size and connection bandwidth are tested (i.e. job types: $Identical$ and $DiffSize$ and node types: H and B). Figure 5.55 shows obtained MRT in four cases. Shown results are for Random underlay graph of average degree 16, Rumor spreading as resource discovery, cache size 16, and HLM load balancing strategy. Result of other configuration produce same rate of difference in MRT values.

Figure explains bandwidth has no effect on MRT when job size is identical (type S). That is high bandwidth is not used because at least one cycle is considered for migrating any job and all jobs have same size. MRT is better when both job size and bandwidth are variable. The reason is for large job size, there is a probability of having two ends of transmission with high bandwidth which is profitable.

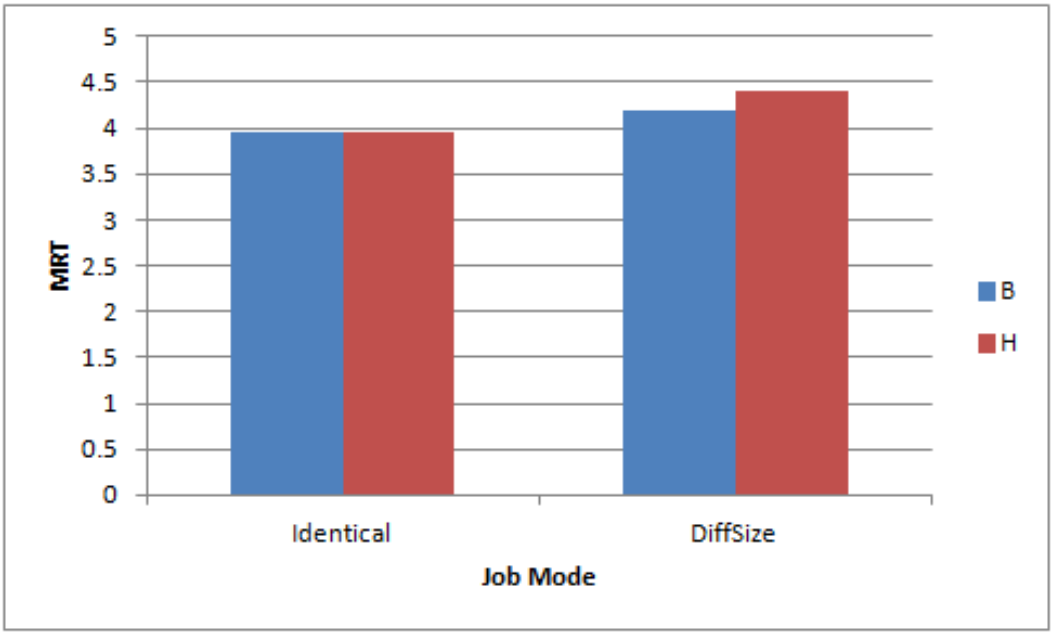


Figure 5.55: MRT for node types (H, B) and job types $(Identical, DiffSize)$

5.2.8 Analyzing of migration graphs

Whenever load balancing is used, a migration graph is generated and stored aside. Each one of above experiments produces a migration graph. Migration graph is directed. One or more migration events between two nodes are represented by an arc. An arc has two attributes: counter and pheromone, see 3.5.

Many migration graphs have been selected for analysis. Only graphs produced from experiments that show high stability are selected. They use $TTL = 3$, local scheme and rumor spreading discovery methods. All selected experiments use underlay network of average degree 16 and cache size (if applicable) equal to 16. Four overlay structures are considered for rumor spreading resource discovery, which are: recentness, reinforcement, activeness, and complex network. Mean value of each indicator is taken as a global threshold. It is applied on each graph.

obtained result are summarized in figures 5.56, 5.57, 5.58, 5.59, 5.60, 5.61, 5.62, and 5.63.

Migration graphs are similar for both indicators because of static arrival rate. Migration graphs of local scheme is a sub graph from the underlay graph because migrations take place strictly between direct neighbors and not any neighbor is concerned by migrations in two directions.

All migration graphs that are produced for rumor spreading are composed of one weakly connected component that contains all nodes from the underlay network. That means all nodes participate in load balancing process. Hence, load may transit in intermediately loaded nodes.

The number of arcs explains the number of distinct pair (source and destination for each direction) that made at least one migration. HLM strategy makes less arcs

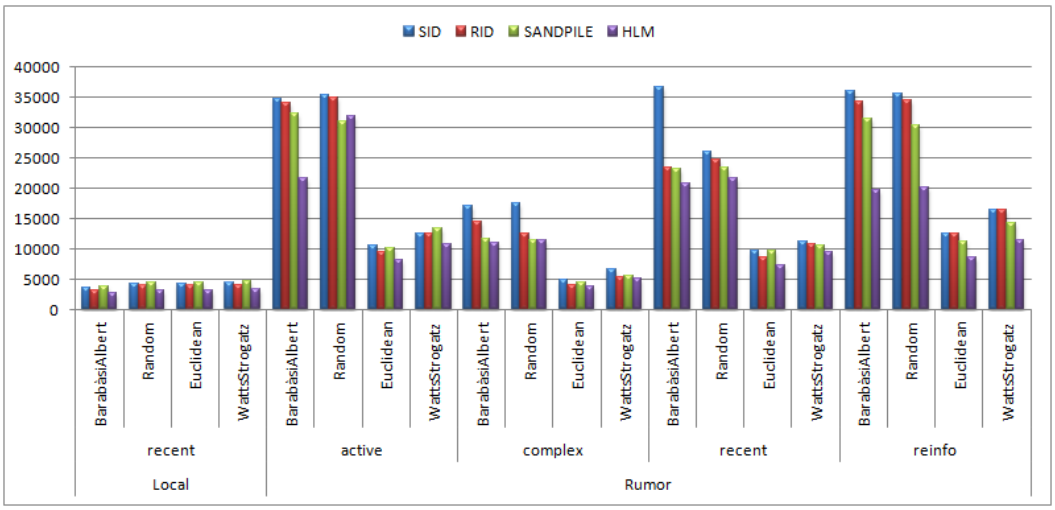


Figure 5.56: Number of arcs in migration graphs resulting from different configurations

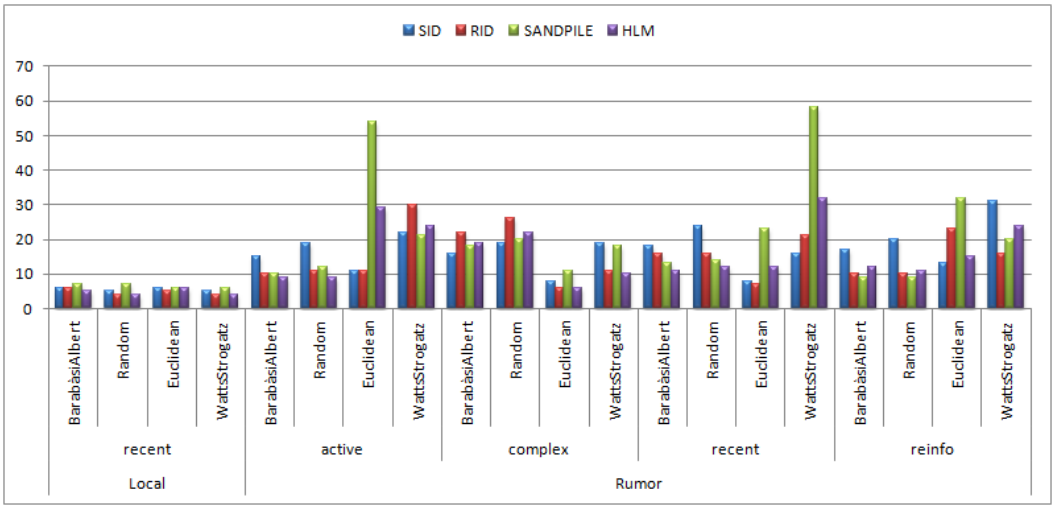


Figure 5.57: Diameter of migration graphs resulting from different configurations

because each underloaded node makes migrations mostly with same overloaded (with a good performance), see figure 5.56.

The graph diameter expresses the distance that load may flow (not necessary for same job) between nodes. For example, at cycle t_1, the job j_i is moved from node u to node v, and at another cycle t_2, another job j_j is moved from node v to node w. Hence, it is said that the load moves for a distance of two edges from u to w. Figure 5.57 shows the diameter of each migration graph. Local scheme produces migration graph with small diameter, which is less than the diameter of the underlay graph. Sandpile creates migration graphs with very large diameter in two cases: activeness on random Euclidean graph and recentness on Watts-Strogatz graph. Both underlay networks are characterized by high clustering coefficient. Sandpile chooses destinations randomly. Hence, jobs may transit in the intermediately loaded

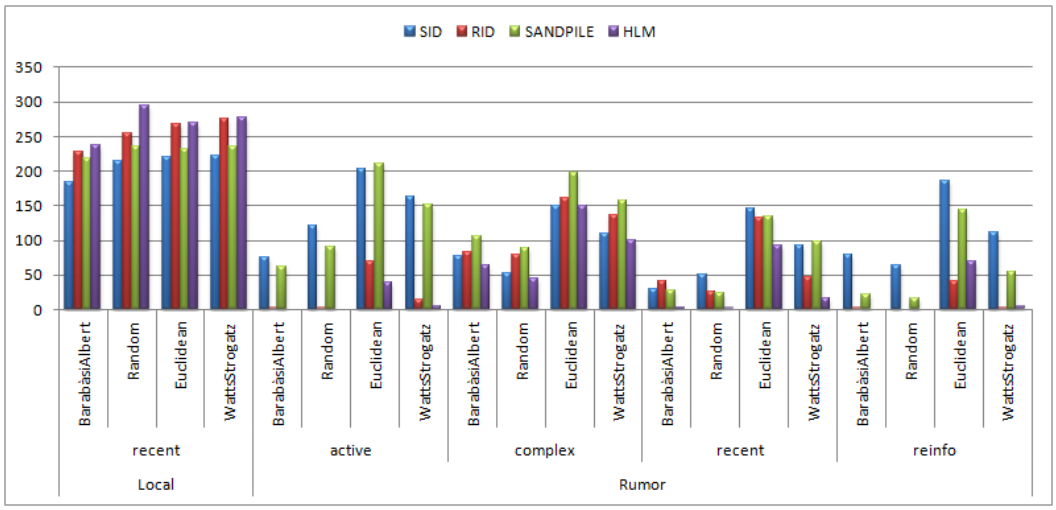


Figure 5.58: Number of "Sources" in migration graphs resulting from different configurations

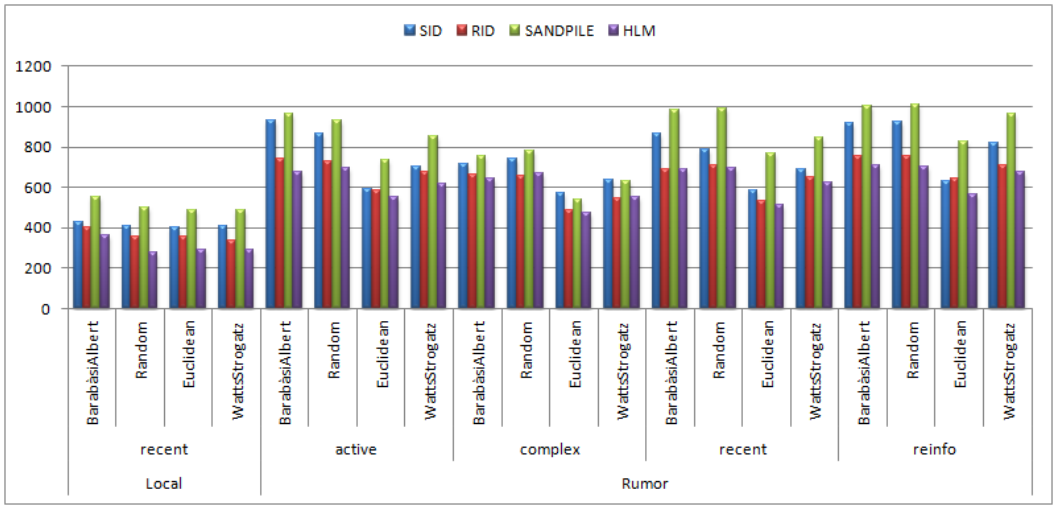


Figure 5.59: Number of "Bridges" of migration graphs resulting from different configurations

nodes and consequentially the diameter of the migration graph is increased.

Sources explain number of nodes that have never been underloaded. Few sources in all graphs (see for example HLM) means there are a few sender only nodes i.e most nodes receive load from others, see figure 5.58.

Bridges are nodes that demand load in some cycles and respond for load demands at others. Number of bridges shows a fluctuation in node state. Figure 5.59 shows high number of bridges in all graphs. Because of Poisson arrival distribution, it is probable that nodes do not receive jobs for several iteration even for those with high arrival rate. Hence, when such nodes send their load to underloaded nodes, they found themselves underloaded in next iterations and may re-fetch jobs from

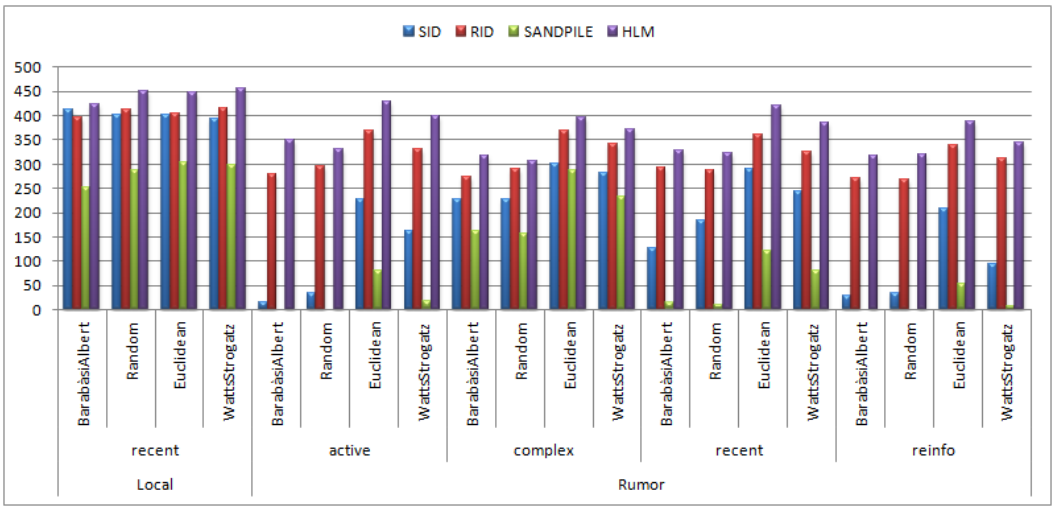


Figure 5.60: Number of "Sinks" of migration graphs resulting from different configurations

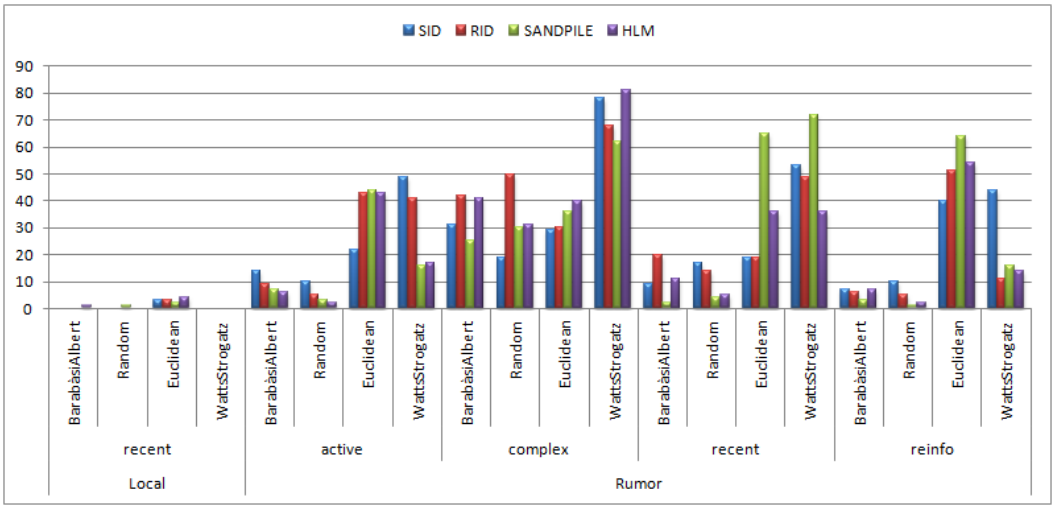


Figure 5.61: Number of "SCC" of migration graphs resulting from different configurations

other nodes.

Another reason is the dynamicity of overlay network i.e. a node that is considered overloaded in a group of nodes at some time it considered underloaded in another group of nodes.

Figure 5.60 shows the number of Sinks. Sinks express the number of receiver-only nodes, i.e. nodes who never send load to others.

A strongly connected component SCC shows probability of moving load (not necessary for the same job) in a circle, see figure 5.61. SCC size explains the number of nodes in each circles. Figure 5.62 shows number of nodes in the largest SCC. Because the stability of overlay network (same as underlay network), a few SCC's are created for local scheme resource discovery. Hence, load transits a few bridges.

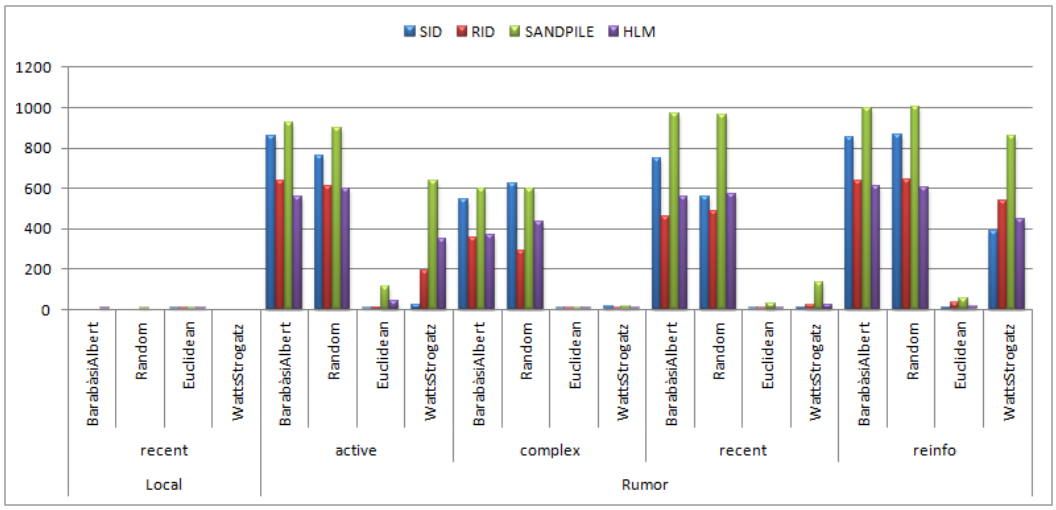


Figure 5.62: Size of largest "SCC" of migration graphs resulting from different configurations

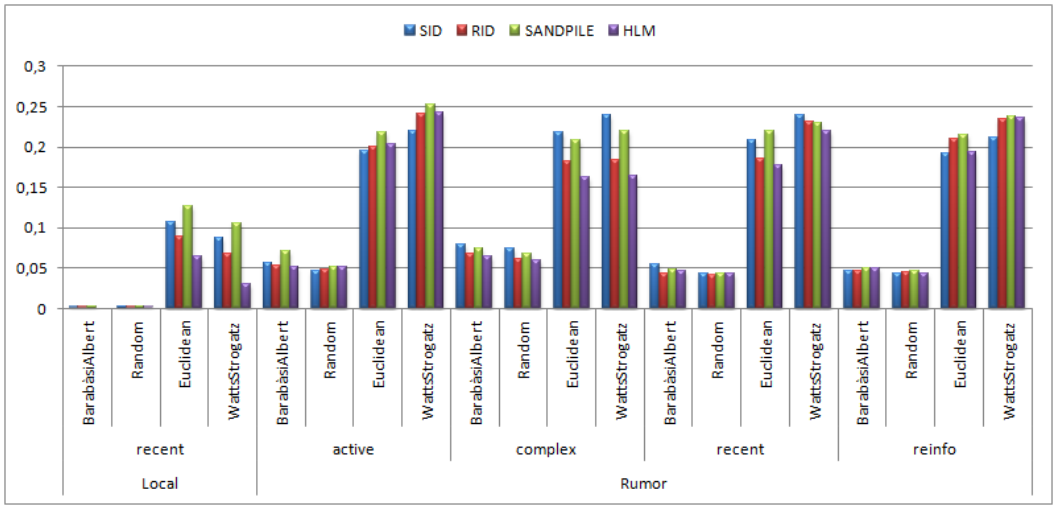


Figure 5.63: Average Clustering Coefficient of migration graphs resulting from different configurations

Average clustering coefficient shows if locality is conserved in the cooperation between nodes. Figure 5.63 shows that Average clustering coefficient of migration graph is approximately same as that of underlay graph.

Some variation in migration graph feature is load strategy dependent. Each of Sender Initiated Diffusion policies (SID and Sandpile) creates more arcs and has less sinks. That is, any node that has load over the average can send load to any underloaded nodes. RID and HLM do the opposite.

In the migration graph, an arc between two nodes is a sign of occurrence of interaction event. High weight value of an arc means frequent communication. The value of the activeness property expresses the impact of a node in load balancing

process.

The large average of out-degree and in-degree in migration graphs is explained by the dynamicity of the overlay network and the variation in job arrival rates.

A node with very high job arrival rate acts as source of jobs. In migration graph, it has high out-degree and zero in-degree.

A node with very low arrival rate acts as sink for jobs. It has high in-degree and zero out-degree.

Nodes with arrival rate near the average play *bridge* roles. They have in-degree and out-degree near the average also. These nodes make the migration graph strongly connected. Nodes are classified as source, bridge or sink after applying a threshold.

5.3 Discussion

5.3.1 Performance of the system

Increasing the density of a network enhances the performance of most strategies except HLM. That is because they rely on stochastic selection of partner nodes to balance load among them while HLM restricts the source selection by the most loaded node.

All strategies have same performance pattern on different network structures at all average degrees.

The diameter and the average degree of underlay network determine the extent of local clusters overlapping. Random, small word, and scale free networks have constantly small diameter which makes load move fast in a network.

The clustering coefficient affects the converging speed of the system to the stability state. It is highly probable that a node makes load balancing with direct neighbors when the clustering coefficient of underlay graph is large. Hence, load moves in local clusters, while random graph is open. A job may move between nodes having a distance more than one.

The study shows that the best performance is obtained in random, Watts-Strogatz, and Barabàsi-Albert networks with an average degree between 16 and 32.

Small average degree of overlay network makes no need to update information for each migration since few nodes can select the same source in each neighborhood. The HLM strategy performs well for an average degree of 16. Then, it needs less time to search for possible load sources and less storage for cached information. Sandpile provides less than good results MRT in all generated graphs. But it asks for very little local computation, especially as it was implemented: each node is considered only once per cycle.

Of course, It is good to distribute load evenly between all system nodes but that costs in both communication and computations. It requires a node to know about the state of all other nodes. Then, a node has to compare its load with that of all others. And, when imbalance is determined, a node has to make many migrations.

Fortunately, distributed policies presented here are able to achieve sufficient load balancing to reach an equilibrium of the complete system.

Mobile agent based broadcasting method produces an overlay network with low density. To increase the density of resulting overlay network, the method needs increasing either the number of active agents or the TTL value. Both actions are not recommended, since both increase the communication amount and especially the latter decreases the accuracy.

In general rumor spreading is faster in term of diffusion speed but it demands more communications.

Rumor spreading is known as simple, robust and efficient broadcasting method. It could provide nodes with a good number of information with high accuracy. The resulting network is connected most of the time. It needs no special configuration environment. The structure of resulting network can be controlled by specifying suitable time-to-live and cache size parameters. $TTL = 3$ keeps the average degree of overlay network between 16 and 32 which is good for all load balancing strategies.

However, the choice of the right discovery method should take into account the amount of communications involved.

For local scheme, it is proportional to the total number of edges in the initial network at each cycle . Unfortunately, the scheme functions quite good for an average degree larger than 16. For rumor spreading, this amount is proportional to $2\times$ number of nodes, and for Mobile Agent based method, it is proportional to the number of mobile agents, number of nodes for the tests. And it can be rather less if needed. Hence each method may be used according to the context, but global scheme ones should be preferred if the initial network has small density or the communication contention is an issue. Again, rumor spreading is better than mobile agent in this context.

5.3.2 Structure of the overlay network

The proposed mechanisms of information management alter the structure of overlay network. A graph with a Poisson distribution is created when only recent information is cached. The same degree distribution with relatively long tail is created when reinforcement is applied. A scale free like degree distribution is created when activeness is used to maintain cached information. The scale free structure gives a better performance. A same result is obtained by [Fukuda 2007b] in the **server deployment**, which is the problem of locating server agents in a network. However, we showed that the structure of the overlay network is self-organized by having highly demanded known by most others instead of manually assigning server agents to hub nodes which becomes useless for dynamic nodes' state.

When the system is heterogeneous in terms of architecture, a complex overlay network is produced. Jobs are moved smoothly in such network since each node sees only compatible nodes.

If nodes have different bandwidths, the migration time has to be considered when computing profitability of moving a selected job. Hence, jobs of small size

are selected for minimizing both transmission time (which maximize MRT) and bandwidth usage.

Interaction modeling is important to visualize and study the behavior of agents. Migration is the main event in the model. Agents take decision of moving some jobs depending on the state of hosting node and its neighborhood. The number of source, bridges and sinks explain the relation between local arrival rate and the role that a node plays in moving load in the network.

Counter type indicator show the quantity of job flow between nodes. Pheromone indicator explains the persistence of a link between some nodes in given time.

Threshold is a technique used to hide eventual events from migration graph. The number of components relates to the clustering coefficient of the underlying graph.

The structure of the migration graph explains the positive impact of information management on the performance of load balancing strategies in different underlay networks.

5.4 Conclusion

Both information and job distributions are modeled using graphs. The characteristics of these graphs help understanding the system behavior, which emerges from the interaction between system entities.

A steady state is reachable in many cases. The structure of the underlay network has a significant impact on the system convergence to a steady state, especially when a node completely depends on the state of direct neighbors. When a node can know the state of distant nodes as sophisticated discovery methods are used, then collected information need to be efficiently managed.

Sandpile performs better in a network of large degree deviation while HLM is more convenient for random networks, however it outperforms all others in other tested networks.

Many solutions are proposed to filter these information using feedback information. The overlay network is the virtual environment that controls system behavior is produced by the interaction among system entities. Controlling the structure of the overlay network makes it possible to guide the system to an equilibrium and may help keeping it in a steady state.

Chapter 6

CONCLUSIONS AND PERSPECTIVES

Contents

6.1 Conclusions 117
6.2 Perspectives and future work 118

6.1 Conclusions

In this thesis we build a model of a Distributed Computing System using agents and dynamic graphs. The model is simulated using Multi-Agent System tool. The behavior of the model is captured and analyzed then results are interpreted within the framework of complex systems.

The overlay network exhibits a stable small world structure independently of the underlay network. However, when we consider two neighbor nodes in the overlay network, then distance in the underlay network depends on the clustering characteristics of the underlay network: These two nodes will most of the time belong to the sale cluster, if such cluster exist in the underlay network. Otherwise distance might be much higher.

More light has been shaded on how the simple behavior of agents can produce a desired global behavior of the system: Roughly speaking, how nodes organize themselves in the overlay network to optimize system performance. Agents efficiently and distributively manage collected information for allowing fast and reliable access to available resource.

Complex system model helps understanding DCS behavior and improving its performance. It describes the way by which entities interact into different levels. Maintaining the structure of overlay network using feedback and self-organization concepts of complex system enhances system performance.

In fact, the set of parameters is very large. Hence, more simplification of the assumptions may be required for further investigation and analyzing.

6.2 Perspectives and future work

The work of the thesis can be extended by considering other parameters. The future work in the subject may includes:

- Dynamizing the parameters of the DCS, like underlay network, bandwidth, etc.
- More information can be mined from the migration graph.
- Applying reinforcement and activeness model on simple agent society like mobile robotics or drones.
- Finally, further development for the simulator by integrating more capabilities can create a valuable research tool.

Indeed, these topics are possible projects for further collaboration between Le Havre University and Babylon University.

Appendix A

WORKLOAD TRACES

A.1 Workload Archeives

A.1.1 Archive list

A non exhausted list of traces of real workload logs is given below:

- Parallel Workloads Archive: A page points to detailed workload logs collected from large scale parallel systems in production use in various places around the world`http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`.
- Real-life based complex Grid workloads collected by Dalibor Klusáček `http://www.fi.muni.cz/~xklusac/index.php?page=meta2009`.
- The Grid Workloads Archive `http://gwa.ewi.tudelft.nl/datasets/`

Logs of real systems use Standard Workload Format file. This format store each workload item (job) in a line in a text file. A line contains eighteen fields that are related to one job. below we list these fields with their description as they found in Parallel Workloads Archive web page.

A.1.2 Log format

The data file is organized according to .SWF file format as it appear in the `http://www.cs.huji.ac.il/labs/parallel/workload/swf.html` web page accessed on end of 2014. The columns of the text files are in the following order:

1. **Job Number** – a counter field, starting from 1.
2. **Submit Time** – in seconds. The earliest time the log refers to is zero, and is usually the submittal time of the first job. The lines in the log are sorted by ascending submittal times. It makes sense for jobs to also be numbered in this order.
3. **Wait Time** – in seconds. The difference between the job's submit time and the time at which it actually began to run. Naturally, this is only relevant to real logs, not to models.
4. **Run Time** – in seconds. The wall clock time the job was running (end time minus start time). We decided to use "wait time" and "run time" instead

of the equivalent "start time" and "end time" because they are directly attributable to the scheduler and application, and are more suitable for models where only the run time is relevant. Note that when values are rounded to an integral number of seconds (as often happens in logs) a run time of 0 is possible and means the job ran for less than 0.5 seconds. On the other hand it is permissible to use floating point values for time fields.

5. **Number of Allocated Processors** – an integer. In most cases this is also the number of processors the job uses; if the job does not use all of them, we typically don't know about it.

6. **Average CPU Time Used** – both user and system, in seconds. This is the average over all processors of the CPU time used, and may therefore be smaller than the wall clock runtime. If a log contains the total CPU time used by all the processors, it is divided by the number of allocated processors to derive the average.

7. **Used Memory** – in kilobytes. This is again the average per processor.

8. **Requested Number of Processors**.

9. **Requested Time.** This can be either run time (measured in wall clock seconds), or average CPU time per processor (also in seconds) – the exact meaning is determined by a header comment. In many logs this field is used for the user run time estimate (or upper bound) used in back-filling. If a log contains a request for total CPU time, it is divided by the number of requested processors.

10. **Requested Memory** (again kilobytes per processor).

11. **Status** 1 if the job was completed, 0 if it failed, and 5 if canceled. If information about chekcpointing or swapping is included, other values are also possible. See usage note below. This field is meaningless for models, so would be -1.

12. **User ID** – a natural number, between one and the number of different users.

13. **Group ID** – a natural number, between one and the number of different groups. Some systems control resource usage by groups rather than by individual users.

14. **Executable (Application) Number** – a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment.

15. **Queue Number** – a natural number, between one and the number of different queues in the system. The nature of the system's queues should be

explained in a header comment. This field is where batch and interactive jobs should be differentiated: we suggest the convention of denoting interactive jobs by 0.

16. **Partition Number** – a natural number, between one and the number of different partitions in the systems. The nature of the system's partitions should be explained in a header comment. For example, it is possible to use partition numbers to identify which machine in a cluster was used.

17. **Preceding Job Number** – this is the number of a previous job in the workload, such that the current job can only start after the termination of this preceding job. Together with the next field, this allows the workload to include feedback as described below.

18. **Think Time from Preceding Job** – this is the number of seconds that should elapse between the termination of the preceding job and the submittal of this one.

This format is extended by Grid Workload Archive .GWA. New columns that concern running jobs on different grid cites are added. The full description of this format is available at http://gwa.ewi.tudelft.nl/grid-workload-format/.

Bibliography

[Acker 2007] D.S. Acker and S. Kulkarni. *A dynamic load dispersion algorithm for load-balancing in a heterogeneous grid system.* In Sarnoff Symposium, 2007 IEEE, pages 1–5, April 2007. (Cited on pages 66, 67, 68 and 71.)

[Bak 1987] Per Bak, Chao Tang and Kurt Wiesenfeld. *Self-organized criticality: An explanation of the* 1/f *noise.* Phys. Rev. Lett., vol. 59, pages 381–384, Jul 1987. (Cited on pages 15 and 22.)

[Barabási 1999] Albert-László Barabási and Réka Albert. *Emergence of Scaling in Random Networks.* Science, vol. 286, no. 5439, pages 509–512, October 1999. (Cited on page 13.)

[Bertelle 2007] Cyrille Bertelle, A Dutot, F Guinand and Damien Olivier. *Organization detection for dynamic load balancing in individual-based simulations.* Multiagent and Grid Systems, pages 1–40, 2007. (Cited on page 1.)

[Biggs 1986] N. Biggs, E. K. Lloyd and R. J. Wilson. Graph theory, 1736-1936. Clarendon Press, New York, NY, USA, 1986. (Cited on page 11.)

[Blum 2005] Christian Blum. *Ant colony optimization : Introduction and recent trends.* Physics of Life Reviews, vol. 2, pages 353–373, 2005. (Cited on page 2.)

[Boccara 2004] N Boccara. Modeling complex systems. Springer, 2004. (Cited on pages 2, 27, 28 and 29.)

[Bonabeau 1997] Eric Bonabeau, Guy Theraulaz, Jean-Louls Deneubourg, Serge Aron and Scott Camazine. *Self-organization in social insects.* Trends in Ecology & Evolution, vol. 12, no. 5, pages 188–193, 1997. (Cited on page 30.)

[Bonabeau 1999a] E Bonabeau, M Dorigo and G Theraulaz. Swarm intelligence: from natural to artificial systems. Oxford University Press, 1999. (Cited on pages 2 and 28.)

[Bonabeau 1999b] Eric Bonabeau, Marco Dorigo and Guy Theraulaz. Swarm intelligence: from natural to artificial systems. Numeéro 1. Oxford university press, 1999. (Cited on page 30.)

[Borko Furht 2010] Armando Escalante Borko Furht, editeur. Handbook of cloud computing. springer, 2010. (Cited on page 10.)

[Braun 2005] Peter Braun and Wilhelm Rossak. *Chapter 2 - From Client-Server to Mobile Agents.* In Peter BraunWilhelm Rossak, editeur, Mobile Agents, pages 7 – 31. Morgan Kaufmann, San Francisco, 2005. (Cited on page 20.)

[Buchowski 1999] Michal Buchowski. *Reclaiming a Scientific Anthropology*. Anthropos, vol. 94, no. 4/6, pages pp. 616–618, 1999. (Cited on page 27.)

[Buyya 2002] Rajkumar Buyya and Manzur Murshed. *GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing*. Concurrency and Computation: Practice and Experience, vol. 14, no. 13-15, pages 1175–1220, 2002. (Cited on page 26.)

[CADIA] CADIA. *Cooperative association for internet data analysis*. http://www.caida.org/. Accessed: 2014-10-15. (Cited on pages 49 and 68.)

[CADIA2007] CADIA2007. *The CAIDA AS Relationships Dataset, September 17 2007*. http://www.caida.org/data/as-relationships/. Accessed: 2014-10-15. (Cited on page 68.)

[Cao 2003] Jiannong Cao, Y Sun, Xianbin Wang and SK Das. *Scalable load balancing on distributed web servers using mobile agents*. Journal of Parallel and Distributed computing, vol. 63, pages 996–1005, 2003. (Cited on pages 21 and 23.)

[Casanova 2014] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson and Frédéric Suter. *Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms*. Journal of Parallel and Distributed Computing, vol. 74, no. 10, pages 2899–2917, June 2014. (Cited on page 26.)

[Censor Hillel 2010] Keren Censor Hillel and Hadas Shachnai. *Partial Information Spreading with Application to Distributed Maximum Coverage*. In Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10, pages 161–170, New York, NY, USA, 2010. ACM. (Cited on page 20.)

[Charpentier 2005] Michel Charpentier. *Cooperative mobile agents to gather global information*. Network Computing and Applications, pages 271–274, 2005. (Cited on page 20.)

[Chau 2003] S.-C. Chau and Ada Wai-Chee Fu. *Load balancing between computing clusters*. In Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on, pages 548–551, 2003. (Cited on page 21.)

[Chierichetti 2010] Flavio Chierichetti, Silvio Lattanzi and Alessandro Panconesi. *Rumour Spreading and Graph Conductance*. In Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10, pages 1657–1663, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. (Cited on page 20.)

[Chopard 2002] Bastien Chopard, Alexandre Dupuis, Alexandre Masselot and Pascal Luthi. *Cellular Automata and Lattice Boltzmann Techniques: An Approach to Model and Simulate Complex Systems.* Advances in Complex Systems, vol. 05, no. 02n03, pages 103–246, 2002. (Cited on pages 29 and 30.)

[Clementi 2013] Andrea E. F. Clementi, Pierluigi Crescenzi, Carola Doerr, Pierre Fraigniaud, Marco Isopi, Alessandro Panconesi, Francesco Pasquale and Riccardo Silvestri. *Rumor Spreading in Random Evolving Graphs.* CoRR, vol. abs/1302.3828, 2013. (Cited on page 20.)

[Coulouris 2011] George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair. Distributed systems: Concepts and design. Addison-Wesley Publishing Company, USA, 5th édition, 2011. (Cited on pages 6, 10, 17 and 43.)

[CRIHAN] CRIHAN. *Computer Resource Centre of Upper Normandy, "Centre de Ressources Informatiques de Haute-Normandie".* http://www.crihan.fr. Accessed: 2014-10-15. (Cited on page 7.)

[Daskalakis 2015] Constantinos Daskalakis, Ilias Diakonikolas and RoccoA. Servedio. *Learning Poisson Binomial Distributions.* Algorithmica, vol. 72, no. 1, pages 316–357, 2015. (Cited on page 39.)

[Demers 1987] Alan Demers, Dan Greene and Carl Hauser. *Epidemic algorithms for replicated database maintenance.* Proceedings of the sixth ..., 1987. (Cited on page 20.)

[Demetrescu 2010] Camil Demetrescu, David Eppstein, Zvi Galil and Giuseppe F. Italiano. *Dynamic Graph Algorithms.* In Mikhail J. Atallah and Marina Blanton, editeurs, Algorithms and Theory of Computation Handbook, pages 9–9. Chapman & Hall/CRC, 2010. (Cited on page 31.)

[DIET] DIET. *Distributed Interactive Engineering Toolbox.* http://graal.ens-lyon.fr/diet/. Accessed: 2014-10-15. (Cited on page 8.)

[Dorigo 2006] M. Dorigo, M. Birattari and T. Stutzle. *Ant colony optimization.* Computational Intelligence Magazine, IEEE, vol. 1, no. 4, pages 28–39, Nov 2006. (Cited on pages 2, 19 and 30.)

[Dorogovtsev 2002] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of networks.* Advances in Physics, vol. 51, pages 1079–1187, June 2002. (Cited on page 13.)

[Dunne 2001] Cameron Ross Dunne. *Using mobile agents for network resource discovery in peer-to-peer networks.* ACM, vol. 1, no. 212, pages 1–9, 2001. (Cited on page 20.)

[EMI] EMI. *European Middleware Initiative.* http://www.eu-emi.eu/. Accessed: 2014-10-15. (Cited on page 8.)

[Érdi 2008] P Érdi. Complexity explained. Springer, 2008. (Cited on page 29.)

[Erdös 1959] P. Erdös and A. Rényi. *On random graphs, I.* Publicationes Mathematicae (Debrecen), vol. 6, pages 290–297, 1959. (Cited on page 15.)

[Feitelson 2015] Dror G Feitelson. Workload modeling for computer systems performance evaluation. Cambridge University Press, 2015. (Cited on page 39.)

[Feldhaus 2012] Florian Feldhaus, Stefan Freitag and Chaker El Amrani. State-of-the-art technologies for large-scale computing, pages 1–17. John Wiley & Sons, Inc., 2012. (Cited on page 8.)

[Ferber 1999] Jacques Ferber. Multi-agent systems: An introduction to distributed artificial intelligence. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st édition, 1999. (Cited on page 25.)

[Forestiero 2007] Agostino Forestiero. *Antares: an ant-inspired P2P information system for a self-structured grid.* Jornal of Network, Information, 2007. (Cited on pages 19, 66 and 68.)

[Foster 2003] Ian Foster and Carl Kesselman. The grid 2: Blueprint for a new computing infrastructure. Elsevier, 2003. (Cited on page 8.)

[Foster 2004] Ian Foster and Carl Kesselman. *Chapter 4 - Concepts and Architecture.* In Ian FosterCarl Kesselman, editeur, The Grid 2 (2), The Morgan Kaufmann Series in Computer Architecture and Design, pages 37 – 63. Morgan Kaufmann, Burlington, 2 édition, 2004. (Cited on page 8.)

[Fountoulakis 2010] Nikolaos Fountoulakis and Konstantinos Panagiotou. *Rumor Spreading on Random Regular Graphs and Expanders.* In Maria Serna, Ronen Shaltiel, Klaus Jansen and Jos Rolim, editeurs, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, volume 6302 of *Lecture Notes in Computer Science*, pages 560–573. Springer Berlin Heidelberg, 2010. (Cited on page 20.)

[Fukuda 2007a] Kensuke Fukuda, Toshio Hirotsu, Satoshi Kurihara, Osamu Akashi, Shin-ya Sato and Toshiharu Sugawara. *The Impact of Network Model on Performance of Load-balancing.* In Akira Namatame, Satoshi Kurihara and Hideyuki Nakashima, editeurs, Emergent Intelligence of Networked Agents, volume 56 of *Studies in Computational Intelligence*, pages 23–37. Springer Berlin Heidelberg, 2007. (Cited on pages 16, 22, 66, 67, 68 and 82.)

[Fukuda 2007b] Munehiro Fukuda and Cuong Ngo. *Resource management and monitoring in AgentTeamwork grid computing middleware.* Computers and Signal processing, pages 145–148, 2007. (Cited on page 115.)

[Giakkoupis 2011] George Giakkoupis. *Tight bounds for rumor spreading in graphs of a given conductance*. In Thomas Schwentick and Christoph Dürr, editeurs, STACS, volume 9 of *LIPIcs*, pages 57–68. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. (Cited on page 20.)

[Giakkoupis 2014] George Giakkoupis, Thomas Sauerwald and Alexandre Stauffer. *Randomized Rumor Spreading in Dynamic Graphs*. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt and Elias Koutsoupias, editeurs, Automata, Languages, and Programming, volume 8573 of *Lecture Notes in Computer Science*, pages 495–507. Springer Berlin Heidelberg, 2014. (Cited on page 20.)

[Gilbert 1959] E. N. Gilbert. *Random Graphs*. The Annals of Mathematical Statistics, vol. 30, no. 4, pages 1141–1144, 12 1959. (Cited on page 15.)

[Globus] Globus. *Globus Toolkit*. http://toolkit.globus.org/toolkit/. Accessed: 2014-10-15. (Cited on page 8.)

[GraphStream] GraphStream. *GraphStream: A Dynamic Graph Library*. http://http://graphstream-project.org/. Accessed: 2014-10-15. (Cited on pages 26 and 68.)

[GridWay] GridWay. *GridWay Metascheduler*. http://www.gridway.org/doku.php. Accessed: 2014-10-15. (Cited on page 8.)

[Gross 2013] Jonathan L. Gross, Jay Yellen and Ping Zhang. Handbook of graph theory, second edition. Chapman & Hall/CRC, 2nd édition, 2013. (Cited on page 11.)

[GWA] GWA. *The Grid Workloads Archive*. http://gwa.ewi.tudelft.nl/. Accessed: 2014-10-15. (Cited on pages 18 and 62.)

[Haeupler 2012] Bernhard Haeupler, Gopal Pandurangan, David Peleg, Rajmohan Rajaraman and Zhifeng Sun. *Discovery through Gossip*. CoRR, vol. abs/1202.2092, 2012. (Cited on page 20.)

[Harchol-Balter 1999] M Harchol-Balter, T Leighton and D Lewin. *Resource discovery in distributed networks*. Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, May 04-06, Atlanta, Georgia, United States, pages 229–237, 1999. (Cited on page 19.)

[Hawkinson 1996] J. Hawkinson. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*, 1996. (Cited on page 67.)

[Hoekstra 2010a] Alfons G. Hoekstra, Ji ří Kroc and Peter M.A. Sloot. *Introduction to Modeling of Complex Systems Using Cellular Automata*. In Jiri Kroc, Peter M.A. Sloot and Alfons G. Hoekstra, editeurs, Simulating Complex Systems by Cellular Automata, Understanding Complex Systems, pages 1–16. Springer Berlin Heidelberg, 2010. (Cited on pages vii and 28.)

[Hoekstra 2010b] AlfonsG. Hoekstra, Ji Kroc and PeterM.A. Sloot. *Introduction to Modeling of Complex Systems Using Cellular Automata.* In Jiri Kroc, Peter M.A. Sloot and Alfons G. Hoekstra, editeurs, Simulating Complex Systems by Cellular Automata, Understanding Complex Systems, pages 1–16. Springer Berlin Heidelberg, 2010. (Cited on page 29.)

[Kshemkalyani 2008] Ajay D. Kshemkalyani and Mukesh Singhal. *Peer-to-peer computing and overlay graphs.* In Distributed Computing, pages 677–730. Cambridge University Press, 2008. Cambridge Books Online. (Cited on page 31.)

[Laredo 2014] J.L.J. Laredo, P. Bouvry, F. Guinand, B. Dorronsoro and C. Fernandes. *The sandpile scheduler.* Cluster Computing, pages 1–14, 2014. (Cited on pages 15, 22, 24, 54, 66, 67, 68 and 71.)

[Lazinica 2009] Aleksandar Lazinica, editeur. Particle swarm optimization. InTech, January 2009. (Cited on page 2.)

[Lemoine 1989] Austin J. Lemoine. *Waiting Time and Workload in Queues with Periodic Poisson Input.* Journal of Applied Probability, vol. 26, no. 2, pages pp. 390–397, 1989. (Cited on page 18.)

[Leskovec 2005] Jure Leskovec, Jon Kleinberg and Christos Faloutsos. *Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations.* In Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM. (Cited on page 68.)

[Leskovec 2014] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection.* `http://snap.stanford.edu/data`, jun 2014. (Cited on page 68.)

[Li 2005] Keqin Li. *Job scheduling and processor allocation for grid computing on metacomputers.* Journal of Parallel and Distributed Computing, vol. 65, pages 1406–1418, 2005. (Cited on pages 18, 39 and 62.)

[Li 2007] Hui Li, Michael Muskulus and Lex Wolters. *Modeling Job Arrivals in a Data-Intensive Grid.* In Eitan Frachtenberg and Uwe Schwiegelshohn, editeurs, Job Scheduling Strategies for Parallel Processing, volume 4376 of *Lecture Notes in Computer Science*, pages 210–231. Springer Berlin Heidelberg, 2007. (Cited on page 39.)

[Li 2009] Hui Li and Rajkumar Buyya. *Model-based simulation and performance evaluation of grid scheduling strategies.* Future Generation Computer Systems, vol. 25, no. 4, pages 460 – 465, 2009. (Cited on page 18.)

[Little 2007] L.R. Little and A.D. McDonald. *Simulations of agents in social networks harvesting a resource.* Ecological Modelling, vol. 204, no. 34, pages 379 – 386, 2007. (Cited on page 16.)

[Lublin 2003] Uri Lublin and Dror G. Feitelson. *The workload on parallel supercomputers: modeling the characteristics of rigid jobs.* Journal of Parallel and Distributed Computing, vol. 63, no. 11, pages 1105 – 1122, 2003. (Cited on pages 18, 38 and 62.)

[Meshkova 2008] Elena Meshkova, Janne Riihijärvi, Marina Petrova and Petri Mähönen. *A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks.* Computer networks, vol. 52, pages 2097–2128, 2008. (Cited on pages 1 and 20.)

[Miller 2008] Michael Miller. Cloud computing: Web-based applications that change the way you work and collaborate online. Que publishing, 2008. (Cited on page 10.)

[Nekovee 2007] Maziar Nekovee. *Worm epidemics in wireless ad hoc networks.* New Journal of Physics, vol. 9, no. 6, page 189, 2007. (Cited on page 16.)

[Newman 2003] M. Newman. *The Structure and Function of Complex Networks.* SIAM Review, vol. 45, no. 2, pages 167–256, 2003. (Cited on pages 11, 16 and 43.)

[Newman 2005] Mark E. J. Newman. Random graphs as models of networks, pages 35–68. Wiley VCH Verlag GmbH & Co. KGaA, 2005. (Cited on page 15.)

[NorduGrid 2014] NorduGrid. *Grid Solution for Wide Area Computing and Data Handling.* `http://www.nordugrid.org/arc/about-arc.html`, 2014. Accessed: 2014-10-15. (Cited on page 8.)

[Pigné 2008] Yoann Pigné, Antoine Dutot, Frédéric Guinand and Damien Olivier. *GraphStream: A Tool for bridging the gap between Complex Systems and Dynamic Graphs.* CoRR, vol. abs/0803.2093, 2008. (Cited on pages 26, 31 and 68.)

[Potiron 2013] Katia Potiron, Amal El Fallah Seghrouchni and Patrick Taillibert. *Multi-Agent System Properties.* In From Fault Classification to Fault Tolerance for Multi-Agent Systems, SpringerBriefs in Computer Science, pages 5–10. Springer London, 2013. (Cited on page 25.)

[PWA] PWA. *Log of real parallel workloads from production systems.* `http://www.cs.huji.ac.il/labs/parallel/workload/`. Accessed: 2014-10-15. (Cited on pages 18 and 62.)

[Salman 2014] Mahdi Abed Salman, Cyrille Bertelle and Eric Sanlaville. *A New Load Balancing Strategy for Distributed Computing Systems.* In Proceedings

of the Fourth International Conference on Complex Systems and Applications ICCSA, pages 143–146, 2014. (Cited on page 24.)

[Shahabi 2005] Cyrus Shahabi and Farnoush Banaei-Kashani. *Modelling Peer-to-Peer Data Networks Under Complex System Theory*. In Subhash Bhalla, editeur, Databases in Networked Information Systems, volume 3433 of *Lecture Notes in Computer Science*, pages 238–243. Springer Berlin Heidelberg, 2005. (Cited on page 19.)

[Shen 2014] Xiang-Jun Shen, Lu Liu, Zheng-Jun Zha, Pei-Ying Gu, Zhong-Qiu Jiang, Ji-Ming Chen and John Panneerselvam. *Achieving dynamic load balancing through mobile agents in small world {P2P} networks*. Computer Networks, no. 0, pages –, 2014. (Cited on pages 67 and 68.)

[Tanenbaum 2006] Andrew S. Tanenbaum and Maarten van Steen. Distributed systems: Principles and paradigms (2nd edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. (Cited on page 6.)

[Thain 2005] Douglas Thain, Todd Tannenbaum and Miron Livny. *Distributed computing in practice: the Condor experience: Research Articles*. Concurr. Comput. : Pract. Exper., vol. 17, no. 2-4, pages 323–356, February 2005. (Cited on page 21.)

[Tisue 2004] Seth Tisue and Uri Wilensky. *NetLogo: A simple environment for modeling complexity*. In in International Conference on Complex Systems, pages 16–21, 2004. (Cited on page 26.)

[Van Der Hofstad 2009] Remco Van Der Hofstad. *Random graphs and complex networks*. Available on http://www. win. tue. nl/rhofstad/NotesRGCN. pdf, 2009. (Cited on page 11.)

[Vidal 2007] JM Vidal. *Fundamentals of multiagent systems with netlogo examples*. In Unpublished. 2007. (Cited on page 25.)

[Voulgaris 2005] Spyros Voulgaris, Mrk Jelasity and Maarten Steen. *A Robust and Scalable Peer-to-Peer Gossiping Protocol*. In Gianluca Moro, Claudio Sartori and MunindarP. Singh, editeurs, Agents and Peer-to-Peer Computing, volume 2872 of *Lecture Notes in Computer Science*, pages 47–58. Springer Berlin Heidelberg, 2005. (Cited on page 19.)

[Watts 1998] Duncan J. Watts and Steven H. Strogatz. *Collective dynamics of 'small-world' networks*. Nature, vol. 393, no. 6684, pages 440–442, June 1998. (Cited on page 16.)

[Werstein 2006] Paul Werstein, Hailing Situ and Zhiyi Huang. *Load Balancing in a Cluster Computer*. In Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies,

PDCAT '06, pages 569–577, Washington, DC, USA, 2006. IEEE Computer Society. (Cited on page 22.)

[Wilensky 2014] Uri Wilensky. *NetLogo: a multi-agent programmable modeling environment*. http://ccl.northwestern.edu/netlogo/index.shtml, jun 2014. (Cited on page 26.)

[Wilensky 2015] U. Wilensky and W. Rand. Introduction to agent-based modeling: Modeling natural, social, and engineered complex systems with netlogo. MIT Press, 2015. (Cited on page 26.)

[Willebeek-LeMair 1993] M. H. Willebeek-LeMair and A. P. Reeves. *Strategies for Dynamic Load Balancing on Highly Parallel Computers*. IEEE Trans. Parallel Distrib. Syst., vol. 4, no. 9, pages 979–993, September 1993. (Cited on pages 21, 23 and 24.)

[Wilson 1970] Robin J Wilson. An introduction to graph theory. Pearson Education India, 1970. (Cited on page 11.)

[Wooldridge 2009] Michael Wooldridge. An introduction to multiagent systems. Wiley Publishing, 2nd édition, 2009. (Cited on page 25.)

Efficient resources management in a distributed computer system, modeled as a dynamic complex system

Abstract: Grids and clouds are types of currently widely known distributed computing systems or DCSs. DCSs are complex systems in the sense that their emergent global behavior results from decentralized interaction of its parts and is not guided directly from a central point. In our study, we present a complex system model that efficiently manages the resources of a DCS. The entities of the DCS react to system instability and adjust their environmental conditions for optimizing system performance. The structure of the interaction networks that allow fast and reliable access to available resources is studied and improvements are proposed.

Keywords: Distributed Computing, Complex System, Multi-agent, network structure, interaction, information management, self-organization, adaptive behavior

La gestion efficace des ressources d'un systéme informatique distribué, modélisé comme un système complexe dynamique.

Resumé : Les grilles et les clouds sont deux types aujourd'hui largement répandu de systèmes informatiques distribués (en anglais DCS). Ces DCS sont des systèmes complexes au sens où leur comportement global émergent résulte de l'interaction décentralisée de ses composants, et n'est pas guidée directement de manière centralisée. Dans notre étude, nous présentons un modèle de système complexe qui gère de manière la plus efficace possible les ressources d'un DCS. Les entités d'un DCS réagissent à l'instabilité du système et s'ajustent aux conditions environnementales pour optimiser la performance du système. La structure des réseaux d'interaction qui permettent un accès rapide et sécurisé aux ressources disponibles est étudiée, et des améliorations proposées.