



**EMBEDDING GAMES:
DISTRIBUTED RESOURCE MANAGEMENT
WITH SELFISH USERS**

JORGE MARIO LONDOÑO PELÁEZ

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

**BOSTON
UNIVERSITY**

BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

**EMBEDDING GAMES:
DISTRIBUTED RESOURCE MANAGEMENT
WITH SELFISH USERS**

by

JORGE MARIO LONDOÑO PELÁEZ

B.S., Universidad Pontificia Bolivariana, 1992
M.A., Boston University, 1999

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2010

© Copyright by
JORGE MARIO LONDOÑO PELÁEZ
2010

Approved by

First Reader

Azer Bestavros, PhD
Professor of Computer Science

Second Reader

John Byers, PhD
Associate Professor of Computer Science

Third Reader

Jonathan Appavoo, PhD
Assistant Professor of Computer Science

Acknowledgments

The path that lead to the completion this work has being a long one, full of obstacles and uncertainties. Getting to the end has been possible thanks to the most valuable teachings of all, those of my parents, Victor and Cecilia.

A special debt of gratitude to my advisor Azer Bestavros, for his continuous support and guidance.

My most sincere gratitude to my readers John Byers and Jonathan Appavoo for all their positive feedback.

To all those who accompanied me in this journey, for making it a memorable one.

the individual costs they incur to secure shared resources necessary to support their application QoS or SLA requirements. Trade and Cap is a market-based scheduling and load-balancing mechanism that facilitates the trading of resources when users have a mixture of rigid and fluid jobs, and incentivizes users to behave in ways that result in better load-balancing of shared resources. In addition to developing their analytical underpinnings, this thesis establishes the viability of NetEmbed, Colocation Games, and Trade and Cap by presenting implementation blueprints and experimental results for many variants of these mechanisms.

The results presented in this thesis pave the way for the development of economically-sound resource acquisition and management solutions in two emerging, and increasingly important settings. In pay-as-you-go settings, where pricing is based on usage, this thesis anticipates new service offerings that enable efficient marketplaces in the presence of non-cooperative, selfish agents. In settings where pricing is not a function of usage, this thesis anticipates the development of service offerings that enable trading of usage rights to maximize the utility of a shared infrastructure to its tenants.

Contents

1	Introduction	1
2	Resource Mapping Techniques for Distributed Systems	7
2.1	Taxonomy	7
2.2	This Thesis in Context	11
3	NETEMBED: A Resource Mapping Service for Distributed Systems	15
3.1	The NETEMBED Service Model	16
3.2	The Network Embedding Problem	18
3.3	NETEMBED Mapping Algorithms	20
3.3.1	Exhaustive Search with Constraint Filtering (ECF)	20
3.3.2	Random Walk with Backtracking (RWB)	26
3.3.3	Lazy Neighborhood Search (LNS)	26
3.4	Network and Constraint Representation in NETEMBED	30
3.4.1	Network Representation	31
3.4.2	Constraint Expression Language	32
3.5	Performance Evaluation	34
3.5.1	Experimental Setting	34
3.5.2	Evaluation Using PlanetLab	36
3.5.3	Evaluation Using Synthetic BRITE Topologies	39
3.5.4	Evaluation Using Queries with Regular Topologies	40
3.5.5	Quality of Returned Results	43
3.5.6	Performance Results Relative to Existing Techniques	44

3.6	Related Work	44
3.6.1	Resource Discovery and Allocation for Testbed Platforms . . .	45
3.6.2	Overlay Networks	47
3.6.3	Service Oriented Architecture (SOA)	49
3.7	Summary	51
4	Colocation Games: Sharing Resources by Splitting the Costs	52
4.1	Colocation Games: Definitions	55
4.1.1	The General Colocation Game (GCG):	57
4.1.2	The Process Colocation Game (PCG):	58
4.1.3	The Multidimensional Process Colocation Game (MPCG): . .	60
4.1.4	The Parallel Process Colocation Game (PPCG):	61
4.2	Analytical Results	62
4.3	PCG: Implementation	70
4.3.1	System Architecture	70
4.3.2	Better Response Heuristics	72
4.4	PCG: Experimental Evaluation	77
4.5	Related Word	86
4.5.1	Resource Allocation Using Micro-economic Mechanisms	86
4.5.2	Resource Allocation Using Game-Theoretic Mechanisms	89
4.6	Summary	90
5	The Trade & Cap Marketplace: Incentives for Load-Balancing	92
5.1	Application Scenario for Trade & Cap	95
5.2	Model Definitions	97
5.3	Demand-based Trading for Fluid Traffic	99
5.4	Demand-based Trading for Atomic Tasks	107
5.5	Trade & Cap (T&C): Combining Atomic and Fluid Tasks	117

5.5.1	Problem Formulation	117
5.6	Prototype Implementation	120
5.7	Experimental Evaluation	124
5.7.1	Traces and Trace Pre-Processing	124
5.7.2	Evaluation of the Demand-Based Fluid Trading System	126
5.7.3	Evaluation of the Atomic Trading System	128
5.7.4	Evaluation of Combined Atomic and Fluid Trading	129
5.8	Related Work	133
5.9	Summary	135
6	Conclusion	137
6.1	Summary of Contributions	137
6.2	Future Directions	139
	Bibliography	141
	Curriculum Vitae	151

List of Tables

3.1	Objects available in NETEMBED constraint expressions.	33
5.1	Characteristics of the WAN trace used in the evaluation.	124
5.2	95% utilization resulting from bandwidth trading.	130
5.3	Traffic volume statistics (in MB) with and without T&C	130
5.4	FT gain for various values of R and IT demand.	132

List of Figures

3-1	Architecture of the NETEMBED service, showing its basic components.	17
3-2	Illustration of the permutation tree that defines the search space . . .	21
3-3	Illustration of the mappings from a query to a infrastructure	24
3-4	ECF Algorithm.	25
3-5	RWB Algorithm	27
3-6	Covered, Neighbor and External sets used by LNS algorithm	28
3-7	The LNS Algorithm.	29
3-8	Mean Search Time for PlanetLab subgraphs	37
3-9	Comparison running queries with PlanetLab as hosting infrastructure	38
3-10	Search times for feasible versus infeasible queries	39
3-11	Mean Search time for BRITE topologies	40
3-12	Mean Time to Find First Match in BRITE topologies	40
3-13	Finding Matchings for a Clique	41
3-14	Finding matchings for composites of regular topologies	42
3-15	Probability distribution of the different types of results	44
4-1	The Colocation Game	57
4-2	Examples of Colocation Games with no Nash Equilibrium.	62
4-3	Construction used in proof of theorem 2.	64
4-4	Examples illustrating Price of Anarchy (PoA) for PCG	66
4-5	Worst and best cases for a player's cost	69
4-6	Pseudocode for the DPKP Better Response Algorithm	74

4-7	Pseudocode for the Branch-and-Bound Better-Response Heuristics . . .	76
4-8	Colocation ratio for synthetic workloads.	80
4-9	Colocation ratio for PlanetLab workloads.	81
4-10	Number of moves until NE for synthetic workloads.	82
4-11	Number of moves for PlanetLab workloads.	83
4-12	Number of trials for synthetic workloads.	84
4-13	Number of trials for PlanetLab workloads.	85
4-14	BFS & DFS Median number of Moves – Synthetic dataset	87
4-15	DPKP Median number of Moves – Synthetic dataset	88
5-1	A BW sharing installation.	96
5-2	Example with two users	106
5-3	Transition graphs for a pure strategies game.	111
5-4	Tight example of PoA for unconstrained game	114
5-5	Tight example for the PoA of the Bandwidth Trading Game	115
5-6	Empirical Price-of-Anarchy based on synthetic worksets	117
5-7	Overall T&C Architecture	120
5-8	HTB-based implementation of the traffic policing component	122
5-9	Downstream trace for a subnet of broadband users	126
5-10	Traffic before and after marketplace reallocation	127
5-11	IT vs FT components per user	127
5-12	Utilization over time for IT sessions with various slack values.	129
5-13	Total Traffic (IT+FT) for various slack values.	131
5-14	Traffic allocations for variable R	132
5-15	IT and FT allocations per user for different slack values.	133

List of Abbreviations

CDN	Content Distribution Network
CPE	Customer Premises Equipment
DT	Delay-Tolerant
DAG	Directed Acyclic Graph
DPI	Deep Packet Inspection
DTB	Delay Tolerant Bulk
ECF	Exhaustive Search with Constraint Filtering
FT	Fluid-Traffic
GPS	Generalized Processor Sharing
ISP	Internet Service Provider
IT	Interactive Traffic
KKT	Karush-Kuhn-Tucker
LNS	Lazy Neighborhood Search
NE	Nash Equilibrium
P2P	Peer-to-Peer
PCG	Process Colocation Game
PPCG	Parallel Process Colocation Game
PoA	Price of Anarchy
PQ	Priority Queueing
QoS	Quality of Service
RMS	Resource Management System
RWB	Random Walk with Backtracking
SLA	Service Level Agreement
SOA	Service Oriented Architecture
VoD	Video on Demand
VoIP	Voice over IP
VCG	Vickrey-Clarke-Grooves
PCG	Process Colocation Game
SWF	Social Welfare
WDP	Winner Determination Problem
WFQ	Weighted Fair Queueing

Chapter 1

Introduction

Managing and arbitrating limited shared resources is a fundamental and difficult problem in computer systems. In this thesis we look at three resource sharing scenarios. The first is the expression of distributed application requirements, of the capabilities a shared infrastructure's resources and finding feasible mappings between them using a constraint satisfaction approach. The second is how to coordinate individual user's fractional requirements of shared resources without explicit user cooperation by exploiting a game theoretic approach. The third is how to minimize the maximum utilization of users demand for a shared resource in a temporal window, by providing an incentive system for users to time shift some of their demand in an automated fashion. The three novel results respectively are: 1) a language for specifying application resource requirements and infrastructure capabilities, and three techniques for finding feasible mappings for the applications up to scales in the order of thousands of nodes and edges, and with response times typically in the order of tens of seconds, 2) a game-theoretic model, describing the interactions of users with fractional requirements who share the cost of the resources, and which let us establish the existence and convergence to equilibrium, as well as bounds on its efficiency for some concrete cases; and 3) a market mechanism, under which self-interested agents may trade allocations of a shared resource over time, as to improve their benefit, while at the same time minimizing the maximum load on the shared resource. This mechanism is proven to have equilibria and to converge to an

equilibrium under certain conditions and worst-case bounds on its efficiency are also proven.

Large distributed applications are now commonplace in many fields of computer science. In turn, these applications often rely on a large infrastructure to provide the resources they need. Some examples are datacenter-based systems [BH09], Cloud Computing [Hay08], Grid Computing [FKNT99], various testbed platforms (*e.g.* PlanetLab [PACR03], Emulab/Netbed [WLS⁺02], GENI [GEN10]), and the Internet itself. Managing resources on these systems is challenging not only because of their scale, but also because of the tradeoffs that exist between the many applications and users sharing the resources. For example, some applications may use a service with no performance guarantees, while other applications may need to ensure some minimum performance level. The problem of finding feasible sets of resources that satisfy the constraints of an application is called the *network embedding problem*.

In the first scenario, the *network embedding problem* is posed as a constraint satisfaction problem where the requirements for various applications and the capabilities of the supporting infrastructure must be matched in order to find feasible mappings. These problems may be challenging by themselves. In many cases they have a combinatorial nature and are NP-hard (see for example [WLS⁺02]), thus it is unlikely¹ that there are algorithms that efficiently find the optimal solution. Given this difficulty, a practical problem is to devise heuristic techniques that can find feasible solutions for practical size instances of the problem. A first contribution of this thesis is NETEMBED, a general framework for the specification and solution of network embedding problems. So, for example, finding the nodes and links for an instance of a Distributed Hash Table (DHT) so that the maximum delay and available bandwidth between adjacent nodes meet some constraints, is easily described within this framework. NETEMBED presents: 1) a language for expressing the re-

¹unless P=NP

source requirements of applications, the capabilities of the infrastructure’s resources, and 2) three mapping algorithms that by using constraint satisfaction approaches, specifically *safe-pruning* and *candidate set reordering*, are able to obtain significant reductions on the search space and solve problem instances of practical sizes (in the order of thousands of nodes and links, with running times in the order of tens of seconds). The correctness of these techniques is also proven, and its scalability is then evaluated empirically on simulations using both synthetic and trace-driven workloads. Chapter 3 discusses NETEMBED in detail.

The second scenario considers the case where multiple users solve their own *network embedding problem* and colocate their applications on a common infrastructure. In particular, if the users have control over the choices of resources for their applications, and the resources can be shared between two or more users, then the users will make strategic decisions as to what resources to choose and each user will make its choice guided by its own, presumed self-interested, objective function. This scenario is enabled by the many technologies that make it possible to switch resources with relatively low overheads. For example, networking technologies such as virtual circuit switching, virtual private networks, or application layer overlays, make it very easy to create a “virtual network” on top an existing network. Similarly on the server side, on-demand computing, virtualization and live-migration of virtual machines are technologies that enable users to very easily reconfigure their applications on top of an existing infrastructure. Even more, when the actions of one user affect the other users, an iterative process in which each user adjusts its allocation decisions in response to the actions of the other users, takes place. In this scenario, fundamental questions that arise are the existence of equilibrium conditions, if the iteration process converges to an equilibrium, and – when equilibria exist – how efficient these equilibria are.

Game-theory provides a framework to study these questions. In particular, non-cooperative games [OR94] address the case where agents make their decisions based solely in the current state of the system and their own objective function. The most commonly used notion of equilibrium is the Nash Equilibrium (NE) [Nas51], which establishes that no agent can improve its benefit by unilaterally changing its action. When a NE exists, it need not be a global optimum (with respect to a predefined system metric). In this case, a measure of efficiency is the Price of Anarchy (PoA), defined as the ratio between the value of the system metric at the worst-case NE and its globally optimal value [Pap01]. Previous works have considered the game-theoretic analysis of shared computing infrastructures when the performance, being used as the valuation metric by the users, is a function of the number of users sharing the resource. This kind of game is known as *congestion games* [Ros73]. However, this model does not apply to computing systems that use a reservation mechanism to guarantee the minimum performance level of an application. To illustrate, virtualization technologies offer the possibility of reserving a fraction of the CPU's capacity to a virtual machine, or various networking technologies (*e.g.* ATM, MPLS, IP+RSVP) of reserving a fraction of the capacity of a shared link to a particular application. In cases like these, the performance is no longer the driver of the user's actions. The second contribution of this thesis, a study of *Colocation Games*, addresses this case. In the *Colocation Games* setting, users are able to provision a fraction of the shared resources, and their cost depends on this fraction. This cost-sharing mechanism is motivated in turn by two factors: It is budget-balanced and it satisfies a notion of fairness. This cost-sharing model is known as *Shapley Cost Sharing* [Sha53], and has also been considered in other applications, such as network formation games [ADK⁺04, CR06], and multicast routing [FPS01].

Colocation Games use a game-theoretic framework to answer the fundamental

questions of this second scenario: It is shown that in general, Colocation Games do not necessarily have equilibrium, but we also prove that for some important cases they do. For the cases where equilibria exists, it is also shown that when users follow best or better response dynamics (best if their action is their optimal action, better if their action improves with respect to their current state) Colocation Games converge to an equilibrium. We also show that equilibria are efficient in the sense that they achieve high resource utilizations. Thus, in applications where in overall achieving high utilization is important (for example in a testbed high utilization translates into allocating larger numbers of experiments), Colocation Games provide a blueprint for a practical mechanism for autonomous agents to allocate resources. Chapter 4 presents a detailed analysis of Colocation Games.

The third scenario considers the case where users compete for a shared resource over time: The users decide when to execute their applications. This leads to a scheduling problem, which has been widely studied as from an optimization perspective (*e.g.* [Leu04]). This problem has also been considered as a congestion game, in which users compete to minimize the turn-around time of their own applications [DT09]. Our scenario is different as it considers the case where users have mixtures of applications: some applications are time-sensitive, while others are delay-tolerant. Time-sensitive applications are those characterized by a step utility function [She95] – if a task is completed within an application-dependent time-frame, it is useful, and otherwise it is useless. For example in networking applications, interactive tasks like browsing, chatting, and audio/video streaming are very sensitive to response times. On the other hand, delay-tolerant applications are those that possess some flexibility with regard to their execution schedule [LR08]. Examples of delay-tolerant applications are network backups, Peer-to-Peer (P2P) downloads, and bulk data transfers. The mixture of both classes of traffic causes problems for time-sensitive applications.

The higher the utilization of a shared link, the longer the response times. This is a common problem in environments such as a campus network, an enterprise network or an Internet Service Provider (ISP), where a large number of users share a common link. The challenge is to devise a mechanism that helps schedule delay-tolerant tasks when users are non-cooperative. Non-cooperative users may not reveal what components of their tasks are delay-tolerant or may go as far as hiding their delay-tolerant tasks by obfuscating the traffic associated with these applications.

The third contribution this thesis is the Trade & Cap (T&C) marketplace. T&C is a mechanism where the users of a common resource can trade their shares of the resource's capacity, and in doing so, will benefit by improving the performance of their two classes of applications: response times for interactive applications, and aggregated throughput for delay-tolerant applications. Fundamental properties of the T&C marketplace such as the existence of equilibrium, the convergence under best or better response dynamics, and the PoA are proven. The analysis and empirical evaluation of the T&C mechanism shows that it effectively provides an incentive for scheduling delay-tolerant tasks in an environment where users are non-cooperative and that the resulting allocation provides a significant reduction of the maximum utilization of the shared resource. The full details of the T&C mechanism are covered in Chapter 5.

Chapter 2

Resource Mapping Techniques for Distributed Systems

Among the many application domains where the network embedding problem has been studied, there is a set of attributes that makes it possible to establish important commonalities and references. Section 2.1 presents a framework that captures these attributes. Section 2.2 places the contributions of this thesis in context and establishes the relationships with respect to the existing works. This Chapter provides a broad overview of the literature and more detailed references, specific to each of the components are provided in their respective chapters.

2.1 Taxonomy

The main classification axis for the purposes of this thesis is the nature of the participants in the distributed systems. On the one hand, if the agents in the system – the users/applications trying to obtain needed resources – allocate resources according to some pre-established mechanism and independently of the benefit each one receives, it is said that this is a *optimization-oriented mechanism* and the design goal is to create mechanisms that achieve some system-wide objective. On the other hand, if the agents have the possibility of making their own decisions and these decisions depend on the benefit each agent obtains from the system, the agents are called selfish, and their strategic behavior is more appropriately modeled using game theory. The lack of coordination between strategic agents usually results in poor

performance for some system metric of interest. When this is the case, the provider of the resources may be willing to establish a mediation mechanism that enforces some form of limited coordination between the users of the resources. This defines a third category, commonly known as micro-economic mechanisms.

The following taxonomy elaborates on these three main classes of mechanisms:

1. *Optimization-oriented mechanisms:*

(a) Centralized: There is a single agent that makes the allocation decisions for the system. This agent controls all the applications sharing the resources. This agent will determine and enforce allocations that optimize the system's objective function.

(b) Distributed: In this case each one of the agents makes the allocation decisions on its own, but the distributed mechanism guarantees (or attempts to approximate) the optimality of a system metric of interest. In these cases the agents follow (and cannot deviate) a pre-established algorithm designed to achieve the intended goal.

2. *Game-theoretic mechanisms:* In this case each agent has its own utility function. This utility depends not only on its allocation decisions, but also on the choices of the other agents in the system. These scenarios are usually modeled using game theory, whereby the set of actions of each agent is its *strategy space*, and the utility function allows the definition of the *best-response* as the action that maximizes the utility for the agent given the current set of choices made by the other agents. Game-theoretic analysis is typically based on the *rationality assumption* that establishes that an agent will choose the action that maximizes its own utility. There are cases where determining this optimal solution is an NP-Hard problem, for these reason this thesis will also use

the notion of a *better-response* when the agents choose an approximated or heuristic solution of their utility maximization problem.

Game-theoretic mechanisms can be further classified as follows:¹

- (a) *Uncooperative Games* [TV07]: When the agents choose their strategy based solely on the current system state and their utility function. Based on how they choose their strategy, this class is subdivided in:
 - i. Pure-strategies games: When the best-response action is chosen deterministically.
 - ii. Mixed-strategies games: When the actions are chosen probabilistically according to some probability distribution. In mixed strategies the agent's goal is to determine the probability distribution that maximizes its expected utility.

When analyzing a game, one of the most fundamental questions is determining its outcome. Although there are many solution concepts, which lead to various definitions of the outcome, of particular interest from a systems standpoint are those that describe an equilibrium state. In particular, the Nash Equilibrium (NE) is commonly used as it defines an equilibrium between self-interested, non-coordinated agents. A NE is a state where no agent can increase its utility by unilaterally changing its current strategy. It is well known that NE are not in general globally-optimal. A measure of the efficiency at equilibrium is the Price of Anarchy (PoA), the ratio between the worst-case NE and the optimal value of the metric of interest.

- (b) *Cooperative (cost-sharing) Games* [JM07]: In this case it is possible for a set of agents to coordinate their choice of strategies so that the result

¹This decomposition follows the one presented in [NRTV07]

is beneficial for all the members of the set. The solution concept for a cooperative game is called a *core* of the game and it is defined by two properties: 1) There is no subset of agents than can benefit by changing their strategies, and 2) It is budget-balanced, *i.e.* the payments of all the agents account for the total cost of the allocated resources. The core property is much stronger than the NE, and it is rare that a cooperative game has a *core*.

3. *Micro-economic Mechanisms:* Mechanisms such as auctions (in their many forms) and exchange/commodity markets have been widely considered as the mean for efficiently managing resources in distributed systems.

There are many auctions formats (ascending, descending, sealed-bid, single-item, multi-item, combinatorial). The combinatorial auction is particularly interesting for resource management in distributed systems, as the bidding language allows for expressing properties such as the fact that set of items are *complementaries* or *substitutes*. A set of resources is complementary if the value of the set is larger than the sum of the individual values of the items. Two sets of resources are substitutes if either set can perform the function requested by the buyer.

One particularly interesting mechanism for combinatorial auctions is the Vickrey-Clarke-Grooves (VCG) auction [AM04]. It has been proven that VCG is the only mechanism that maximizes the social welfare (the sum of the private values of the winners), and that makes truthful bidding the dominant strategy – no bidder benefits by bidding values other than its true value. It has its limitations too [San96]. It is not budget balanced (the sum of winner bids is not equal to the payments to the owners of the resources), the associated Winner Determination Problem (WDP) is NP-Hard, and it is susceptible to

various attacks, for example collusion between the bidders.

An exchange market allows the interaction of a set of sellers and buyers. The goal of the exchange market is to find a market clearing solution: The set of prices that makes the aggregate demand equal to the aggregate supply of each commodity in the market. A classical result by Arrow-Debreu [AD54] establishes sufficient conditions for the existence of this solution. However, general solutions are known only for the case of linear utility functions [Ye08].

2.2 This Thesis in Context

NETEMBED falls within the category of optimization-oriented mechanisms. In NETEMBED the goal is to find feasible mappings (resource assignments) for a distributed application. Once found, the resource allocation system can evaluate the optimality of the feasible solutions to make the allocation decision. Similar problems have been already considered in the literature, for example in [WLS⁺02], [ALR03], and [OAPV05, AOPV08].

In [WLS⁺02] the authors present *wassign* a resource allocator for Emulab that uses a genetic algorithm. Another resource allocator for Emulab is *assign*, which is described in in [ALR03]. Both systems are designed with the purpose of maximizing the number of experiments that can be instantiated in the infrastructure and a fundamental characteristic of this system is that the allocation decisions have minimal effect for the experiment being allocated. This means that for the users the utility depends only on the fact of being able to execute the experiment or not, and the system objective of maximizing number of experiments maximizes the social value for its users.

SWORD [OAPV05, AOPV08] is a mapping service for PlanetLab. It is designed with the purpose of finding the set of resources that better map the requirements of

the user’s experiment. Giving the hardness of the problem, SWORD uses various pruning heuristics to find feasible solutions and returns the best one among the solutions found.

A problem often found when solving embedding problems with heuristics, the the abovementioned examples, is that they are prone to false negatives – returning a “no-mapping found” when a valid mapping exists. An alternative approach is the use of constraint satisfaction techniques. In fact, this approach has been previously explored by Considine *et al* [CBMp03]. NETEMBED further develops these approaches by introducing two techniques: *safe-pruning* – pruning only unfeasible regions of the search space, and *candidate sets reordering* – constructing candidate sets and ordering the search based on the size of the candidate sets. Both techniques significantly improve the scale of problems that can be handled and are false-negative free.

In Colocation Games various self-interested agents colocate their applications on a shared infrastructure, while sharing the cost of the resources. The agents can choose the resources for their applications from any of the feasible mappings, reserve a fraction of the capacity of the resources to guarantee the minimum performance requirements of the application, and split the cost in proportion to the fraction of the resource allocated. This represents a departure from classical *congestion games*, such as [KP99, CV02, RT02], where there are no performance guarantees, but instead the resulting performance is a function of the number of applications sharing the resource, and agents compete to improve their own performance. In Colocation Games performance is a constraint that is enforced by means of a resource reservation mechanism, and the cost is variable and depends on the reserved fraction of each resource. Our analysis shows that in general, Colocation Games do not to have a NE, but some restricted forms of the Colocation Games do have one or more NE and converge to a NE under better or best-response dynamics. For these restricted cases,

we prove bounds on the PoA, and an experimental evaluation with actual workload traces shows that in practice the efficiency of the Colocation Game is better than the bound, thus making the game a practical mechanism for resource management between selfish agents. As Colocation Games do not have any coordination mechanism – all the agents follow their best or better responses – they are classified as *uncooperative games* according to the framework of the previous section.

The Trade & Cap (T&C) marketplace introduces a mechanism that allows the exchange of “resource shares” between users of a common resource. In particular, the T&C marketplace is presented as a scheduling mechanism where the users have various mixtures of interactive and delay-tolerant tasks. In doing so, the T&C marketplace incentivizes selfish users to schedule the execution of delay-tolerant tasks during periods of low demand, thus balancing the load on the resource over time. This is beneficial for the resource provider when the cost associated with the resource depends on its utilization.² The idea of scheduling delay-tolerant jobs during periods of low utilization was presented by Laoutaris and Rodriguez [LR08]. They propose two approaches to realize this concept. The first is to provide some reward to the agents who postpone the execution of their jobs. The second is to implement a store and forward service across some vantage points on the Internet, so that data can be relayed between those points at periods of low utilization. This second approach is further developed in [LSRS09] to provide the means for the delivery of Delay Tolerant Bulk (DTB) data. This addresses the problem when the resources and the jobs that need to be executed are under the control of a single authority, in which case, the problem is classified as an optimization oriented approach. For the first one though, no concrete mechanism has been proposed yet. The T&C marketplace is a mechanism that uses the first approach. This is particularly important for cases

²A common example of this situation is high capacity network connectivity, which is priced according to the 95/5 rule.

when the resource consumers are non-cooperative, but only interested in maximizing their own utility. By giving them the right incentive, it is possible to improve both the performance of interactive applications, and the peak-to-valley ratio that affects the provisioning and costs of the provider. The market operates as a mediator that enables the exchange of “resource shares” between users, thus the T&C mechanism belongs to the class of micro-economic mechanisms.

Chapter 3

NETEMBED: A Resource Mapping Service for Distributed Systems

An emerging trend in distributed system design is the use of configurable networks, in which network resources may be named, acquired, and manipulated by applications and users in a much more flexible manner than what are possible using current, standard Internet services. Using today’s Internet services, such as DNS, an application is able to resolve (or map) a name to an IP address, but not much more. This chapter presents NETEMBED a mapping service that allows an application to name resources in a much more expressive fashion – for instance by specifying a desirable topological configuration of nodes and links, and by constraining the attributes of such nodes and links.

One canonical example of a configurable network is that of an overlay network of end-systems. Such overlays are increasingly used as targets of choice for new distributed applications, including large-scale data management systems, *e.g.*, distributed data stores [RWE⁺01, SDR04], content delivery networks [BLBS03, BCMR02, CDK⁺03, CRSZ02, KRAV03, KB04], and specialized resource reservation/allocation services supporting the operation of grid computing [LZZ⁺04]. Applications running on overlays require services that allow them to flexibly make content placement decisions, routing decisions, and other decisions regarding resource selection.

Another canonical example of a configurable network is that of network testbeds such as Emulab/Netbed [WLS⁺02] and PlanetLab [PACR03], as well as the envi-

sioned GENI testbed [GEN10]. The development of these network testbeds is motivated by the necessity of having more realistic experimental environments, where new protocols and services can be tested under conditions as close as possible to those found in real networks. Users of (and applications running atop) such testbeds are allowed considerable flexibility in selecting the sets of resources used in conducting a distributed experiment.

3.1 The NETEMBED Service Model

An embedding service could be thought of as a component of a large distributed system that other applications could use as a primitive for the realization of their resource needs. The following are example application scenarios:

- A dynamic multicast service, where an overlay distribution tree must be configured subject to a set of constraints so that some QoS requirements are satisfied.
- A peer-to-peer system, where location, bandwidth and delay of a subset of nodes, such as “directory nodes” (*e.g.*, the nodes of a distributed hash table) play an important role in the performance of the look-up service and/or the overall performance of the whole system.
- A network monitoring system, where the health of the network could be monitored by a subset of nodes satisfying some connectivity and location constraints.
- A sensor network in which it is desirable to locate a subset of sensors that possess certain capabilities and satisfy some resource, location, and/or network connectivity constraints.
- A grid application that needs to allocate a subset of nodes with certain capabilities and some connectivity requirements between them.

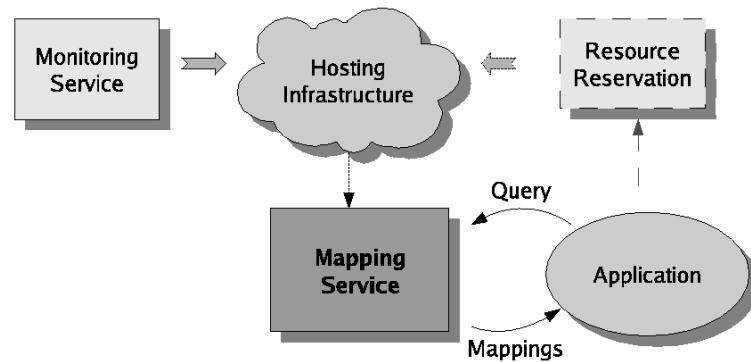


Figure 3-1: Architecture of the NETEMBED service, showing its basic components.

- A Service Oriented Architecture (SOA) application, where the complete application is the result of the composition of services that need to be instantiated across a collection of distributed resources. The requirements of each service and the Quality of Service (QoS) constraints of the application limit the possible choices for allocating the various services.

In such cases, it makes sense to have a resource mapping service where applications can submit their resource requirement specifications, and in response get a list of possible resource assignments. Such a service would involve the key components illustrated in Figure 3-1 and described below:

1. A model of the real network that characterizes the resources available. Such model could be maintained either by a monitoring service, a resource manager, or a combination of both.
2. The mapping service itself, to which applications would submit their queries and get a list of possible mappings. An interactive service would facilitate the adjustment (negotiation) of the requirements if the query cannot be satisfied, or allow it to adjust the mapping dynamically, as the application's needs change. The service can operate in a distributed fashion simply by keeping an up-to-date copy of the model on each server.

3. Optionally, if a resource reservation system is in place, applications would allocate the selected mapping and the network model would be adjusted accordingly.

As described, the NETEMBED service is focused on finding “feasible” mappings as opposed to finding an “optimal” mapping. As noted in Chapter 2, the identification of an objective function depends on the application, and more importantly, on the entity doing the optimization. Nevertheless, incorporating the evaluation of an optimality goal is easily accomplished once the candidates have been found. For these reasons the scope of NETEMBED is limited to the identification of feasible mappings.

3.2 The Network Embedding Problem

This section provides the basic definitions for terms alluded to earlier and present a formal definition of the *Network Embedding Problem*, which is used throughout this thesis.

The term *Hosting Infrastructure* is used when referring to the target of an embedding service. A hosting infrastructure (*e.g.*, Internet or PlanetLab) is described by a graph $R = \langle V, E \rangle$ and a characterization of its links and nodes, which may include measured metrics represented either as numeric values or ranges, and categorical classes such as “*Link (n₁, n₂) is 802.11g*” or “*node n₁ is linux-2.6*”, for example.

The term *Query Network* refers to the network that needs to be embedded into the hosting infrastructure. A query network is given by the graph $Q = \langle V, E \rangle$ and a characterization of its links and nodes, which represent constraints on any feasible embedding. As with the hosting infrastructure, such constraints may be numerical or categorical in nature.

The term *Mapping* is used to refer to a one-to-one (injective) function $m : Q \rightarrow R$, such that for all query network nodes $q \in Q$, $r = m(q) \in R$ is the corresponding

node in the hosting infrastructure, and all node and edge constraints are satisfied by such mapping. To simplify notation, the notation $q \rightarrow r$ is used to indicate that node q maps to node r . In principle, if N_Q is the number of nodes in Q and N_R is the number of nodes of R , then any permutation of N_Q elements of N_R could be a mapping, making the search space very large even for small values of N_Q and N_R .

The term *Constraint Expression* refers to a boolean expression that specifies additional relationships that must be preserved by the mapping function. For example, the user may be interested in a mapping that restricts the average delay between nodes within a percentage range, or that certain particular nodes get mapped to physical nodes with certain attributes, *e.g.* operating system, processor type, speed, *etc.* Such constraints take the form of a boolean expression relating query network nodes/links to hosting infrastructure nodes/links.¹

The *Network Embedding Problem* is the problem of finding a mapping $m : Q \rightarrow R$ that satisfies the constraints established by the *Constraint Expression*. It may be the case that the user is either interested in finding feasible mappings, in finding a mapping that improves his current utility (later on referred to as a *better move*), or in finding the optimal mapping, which maximizes its utility or minimizes its cost.²

An *Embedding Service* is a system that takes as input the description of a hosting infrastructure and the specification of a query network and returns as output a (possibly empty) set of mappings from the latter to the former. Typically, an embedding service would be associated with a single hosting infrastructure that constitutes a “real” infrastructure and would be used by users or applications to facilitate the identification of resources that could be used to instantiate a “virtual” network that satisfies the constraints specified in the query network.

¹NETEMBED uses a simple constraint expression language, which will be described later in Section 3.4.

²The user’s optimal allocation given the current allocations of the other users is called its *best-response*, and will be fundamental in the analysis in the subsequent chapters.

In this work, the terms “hosting infrastructure”, “query network”, and “mapping” are used interchangeably with the terms “real network”, “virtual network”, and “embedding” respectively. The latter set of terms is typically used when referring to an embedding service tied to a specific infrastructure or testbed.

3.3 NETEMBED Mapping Algorithms

This section presents three mapping techniques developed to solve *The Network Embedding Problem* using constraint satisfaction techniques. These techniques exploit the constraints of the problem as the means for pruning the search space and are able to handle practical-size problem instances.

3.3.1 Exhaustive Search with Constraint Filtering (ECF)

A mapping is a N_Q -permutation of the N_R nodes in the infrastructure network. An exhaustive search would explore each one of the $P_{N_R}^{N_Q}$ permutations to find feasible ones. Naïvely doing so is impractical even for relatively small values of N_R and N_Q . ECF organizes the search space in the form of a *permutations tree* constructed in the following manner:

1. From the root, there is a branch to each one of the possible mappings of the first query node.
2. The children of the nodes at the i^{th} level are the possible mappings for the $i+1$ query node, given the assignments in the previous levels.

Figure 3.2 illustrates this construction.

ECF is a search on this tree. It could be implemented either as a Depth-First Search (DFS) or a Breath-First Search (BFS). Noticing that the amount of state

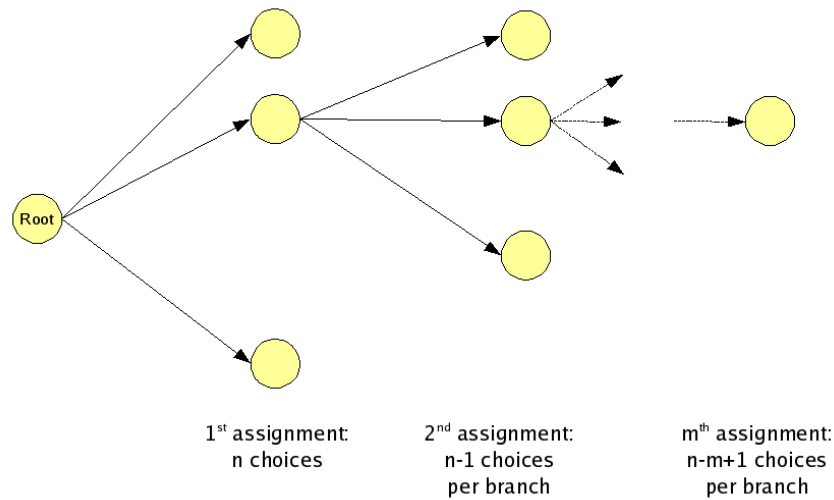


Figure 3·2: Illustration of the permutation tree that defines the search space

required by BFS may be exponentially large³, ECF uses a DFS strategy. In fact, by handling the sets of nodes as ordered sets (each node has a numerical identifier) the amount of state information is determined by the current depth, which is at most N_Q . To avoid exploring the whole tree, ECF prunes a branch as soon as it is determined to be infeasible. Observe that once the mapping to a node is determined to be infeasible, the entire subtree rooted at this node is unfeasible and it can be safely pruned.

In addition to the pruning strategy, ECF also does a reordering of the query nodes to minimize the number of nodes visited. A DFS visits every node in the tree once, therefore its performance is determined by the size of the tree. By evaluating node and link constraints, it is possible to determine the sets of candidate mappings for each one of the query nodes. So for example, for query node $q \in Q$ there is a set of feasible candidates $C_q \subset R$ and the set of sets-of-candidates can be ordered by their cardinality, *i.e.* $\{C_1, C_2, \dots\}$ such that $|C_i| < |C_j|$ for all $i < j$.

³BFS keeps the queue of nodes for the next level, which at the last level is of the size of the search space

Lemma 1 establishes that this reordering step minimizes the size of the tree.

Lemma 1. Minimum Size of the Permutations Tree

The total number of nodes in the permutations tree for a search algorithm is minimized if the algorithm examines the query nodes in an ascending order based on the number of candidate mappings for each query node.

Proof. Let n_i be the number of candidate mappings for virtual node v_i . According to the statement in the Lemma, nodes are examined in increasing order of their candidate mappings. Thus, $n_i < n_{i+1}$ for $1 \leq i < N_Q$ and the total number of nodes in the permutations tree is given by

$$S = n_1 + n_1 n_2 + \dots + n_1 n_2 \cdot \dots \cdot n_{N_Q} = n_1(1 + n_2(1 + \dots n_{N_Q-1}(1 + n_{N_Q}) \dots)) \quad (3.1)$$

Assume that there is an alternative ordering of nodes that yields a smaller total number of nodes (namely S') in the permutations tree—*i.e.*, $S' < S$. In particular, without loss of generality, assume that such an ordering would switch the order with which nodes v_1 and v_2 are examined, which yields the following

$$S' = n_2(1 + n_1(1 + \dots n_{N_Q-1}(1 + n_{N_Q})))$$

Since, by definition, $n_2 < n_1$, then

$$S' - S = n_2 - n_1 > 0$$

This implies that the alternative ordering yields a larger total number of nodes—*i.e.*, $S' > S$ —which is a contradiction. Thus, by having n_1 as the first factor minimizes the total number of nodes if the order of the rest of the nodes is kept fixed. The same procedure can also be applied to the successive factors $(1 + \dots n_{N_Q-1}(1 + n_{N_Q}))$ so that each consecutive internal factor is minimized as well. In consequence, ordering the mappings in increasing order by the number of candidates minimizes (3.1). \square

The above Lemma implies that it is advantageous for the degree of nodes closer to the root of the permutations tree to be as small as possible. Thus, by applying the constraint expression the ECF algorithm determines the number of possible mappings for each virtual node and sorts them in increasing order in a list L_S .

In addition to the above observation, note also that given the current assignments for q_1, \dots, q_{i-1} , if q_i has edges with any of its predecessors, the number of choices is reduced even more because these edges have to be preserved.

During the first stage of the ECF algorithm, the constraint expression is applied to each possible pair of virtual and real edges. As illustrated in Figure 3-3, this produces a list of candidate mappings per edge of the form:

$$\{(q_1 \rightarrow r_1, q_2 \rightarrow r_2), \dots\}$$

ECF stores this mapping in a data structure that provides the candidates for the second stage. The data structure is a sparse 3D-matrix F that will be referred to as the *filter matrix*. Each cell in F has coordinates (v, r, v_s) and contains the set of candidate mappings for v_s , when v is mapped into r . Therefore, each edge matching adds one element in two cells⁴:

$$(q_1, r_1, q_2) \leftarrow r_2 \quad (q_2, r_2, q_1) \leftarrow r_1$$

Conversely, when there is no match, ECF keeps these results in a second filter \bar{F} , constructed in exactly the same way. \bar{F} will then be useful in restricting the set of candidates given the current partial mapping.

The ECF algorithm then proceeds as follows:

- (1) Pick the first virtual node v_s from the sorted list L_S , i.e. the one with the least number of candidates. Being the first, it can be chosen from:

$$\bigcup_{all\ v \in N_Q, r \in N_R} F[v, r, v_s] \tag{3.2}$$

⁴For the undirected case. If the network is represented as a directed graph only one of the sets is updated

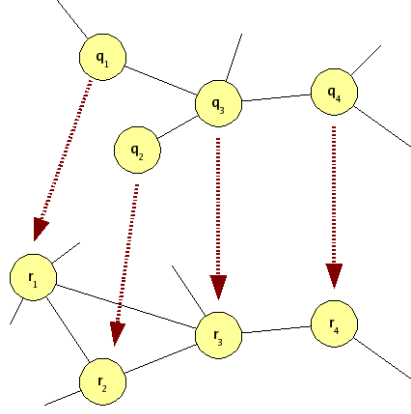


Figure 3-3: Illustration of the mappings from a query to an infrastructure

(2) For subsequent nodes, say node v_i , choose the mappings from the intersection of candidates for all previous nodes v_j that have an edge with v_i and that do not violate any of the constraints. Real nodes that have been already assigned cannot be considered. Expression (3.3) gives the set of candidate nodes⁵. This process guarantees that each additional node mapping is consistent with the topology and the established constraints.

$$\left(\bigcap_{\text{all } j < i | (v_j, v_i) \in E_Q \wedge (v_i, v_j) \in E_Q} F[v_j, r_j, v_i] \right) - \left(\bigcup_{\text{all } j < i | (v_j, v_i) \in E_Q \wedge (v_i, v_j) \in E_Q} \overline{F}[v_j, r_j, v_i] \right) - \{r_1, \dots, r_{i-1}\} \quad (3.3)$$

Once the search reaches the N_Q^{th} level (the DFS reaches a leaf of the tree), then by induction this is a valid mapping for the whole query. The complete algorithm incorporating these filters is given in Figure 3-4.

⁵When the network is represented by a directed graph

```

function beginSearch()
  root ← createRoot
  Candidates ← set of real nodes defined by (3.2)
  call search(root, Candidates)

function search(node, Candidates)
  if node is at depth  $N_Q$ 
    report mapping defined by branch from node to root
    return
  for each c in Candidates
    add c as a child of node
    NextCandidates ← set from filter (3.3)
    if NextCandidates is empty return
    call search(c, NextCandidates)
    remove child c from node

```

Figure 3-4: ECF Algorithm.

Correctness of ECF: ECF does a depth-first traversal of the permutations tree. Clearly, if the search visits the whole tree, any possible mapping would be found. On the other hand, ECF does not visit the whole tree, but prunes a branch as soon as it finds that there are no feasible mappings for the current query node, at which point any child branches also fail to satisfy the constraints for the current node and therefore it does not drop any feasible solutions.

Worst-Case Performance of ECF: In the worst case, the constraints are loose enough so that any node in the hosting infrastructure is a valid candidate for each query node, and any edge as well. The hosting infrastructure in the worst case is a clique (any graph of $N_Q < N_R$ nodes is a subgraph of a clique of size N_R). Under these circumstances both heuristics fail to prune any candidates and ECF will explore the entire permutations tree. Section 3.5 presents the results of an empirical analysis, illustrating that under minimal sets of practical constraints these techniques achieves an almost linear scalability.

3.3.2 Random Walk with Backtracking (RWB)

ECF performs an exhaustive search and retrieves *all* feasible mappings for a given query. However, for many applications, finding *all* feasible embeddings is not necessary. For example an application may just require the identification of *any* (single) feasible embedding, or it may require the exploration of a representative subset of feasible embeddings (*i.e.*, a region of the solution space) in order to optimize resource allocation over that subset. For such applications or in general when obtaining the entire set of candidate embeddings is not necessary, and/or if the problem size is too large to be handled by an exhaustive method, a non-deterministic version of ECF may be more appropriate. This motivates an alternative to the ECF algorithm described below.

The algorithm uses the same filtering conditions (3.2) and (3.3) to randomly choose the next mapping. If at some point the algorithm reaches a dead end, it backtracks to the previous virtual node and selects another candidate. If at some level there are no candidates left, it backtracks to the parent node, and so on. If by backtracking the algorithm reaches the root node, it returns with no solution. By virtue of the randomness with which candidate mappings are selected, and the backtracking-nature of the search for a feasible embedding, this algorithm is referred to as the Random Walk with Backtracking search. The pseudo-code for the algorithm is shown in Figure 3-5.

3.3.3 Lazy Neighborhood Search (LNS)

Both the ECF and RWB algorithms have a potentially problematic attribute: their space requirements may become prohibitive, especially for under-constrained queries. In particular, the worst-case space requirement for the two filter matrices used in both ECF and RWB is $O(n|E_Q||E_R|)$. This case could occur for under-constrained queries

```

function randomWalkSearch()
  current ← first virtual node
  while(current is not null)
    Candidates ← set of real nodes from filter (3.2) or (3.3)
                  minus discarded for current
    selected ← a random element from Candidates
    do until added a child or there are no more candidates
      if selected is compatible with current mapping
        add selected as a child of current
        current ← selected
        if current is the last node,
          return the mapping given by the
            branch from current to the root
        otherwise
          add selected to the list of discarded for current
    if could not find compatible child,
      current ← parent(current) // Backtrack
  return "no solution"

```

Figure 3-5: RWB Algorithm

over dense networks, since most links would be a match of each other and each filter position could map close to n candidates. Roughly speaking, the query size could also be close to the hosting infrastructure, so for dense graphs $|E_Q| \sim |E_R| \sim n^2$, giving a worst-case space of $O(n^5)$, which even for small query networks (say a few dozen nodes) may be too large to handle as it will overrun the system’s memory quickly.⁶

To reduce the space requirement, the algorithm presented in this section seeks to minimize the amount of state information kept during the search, and to making pruning decisions along the way. The main idea behind this algorithm is illustrated in Figure 3-6.

At any point in time there are three sets: *Covered*, which contains the set of

⁶Notice that if the hosting infrastructure is dense (as with overlays, in which there is an overlay link between every two nodes), then the topological constraints implied by the virtual network do not help much in reducing the number of candidate edges. This only gets worse if the query is under-constrained. In the worst-case the graph could be a clique and the set of candidates would be the set of all edges going to the nodes not yet selected. In this situation, the heuristic introduced in Section 3.3.1 of sorting by number of candidates becomes ineffective and it is truly exploring the full permutations tree.

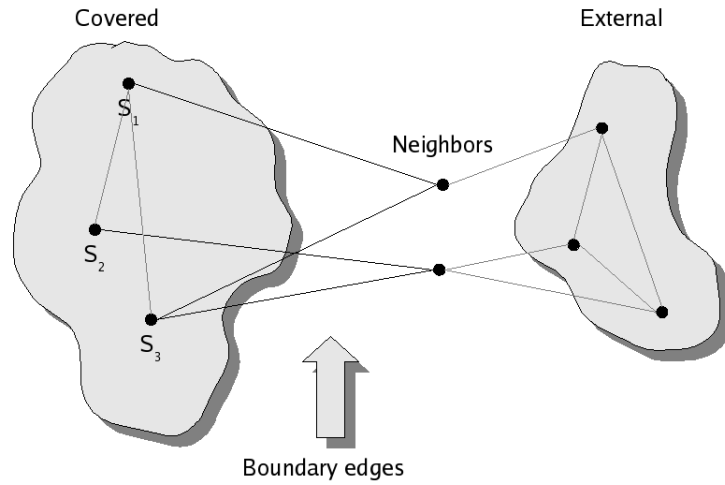


Figure 3-6: Covered, Neighbor and External sets used by LNS algorithm

virtual nodes already matched, *Neighbors*, which contains the set of nodes connected to at least one covered node, and *External*, which contains the set of query nodes with no connections to the covered set. At each stage, the algorithm picks one of the neighbor vertices, and checks if there is a mapping that would allow it to satisfy topological and query constraints against all covered nodes, and if so, it adds that vertex to the covered set. This guarantees that all the covered vertices constitute a valid partial match. Clearly, when all query vertices are in the covered set this is a complete matching. Figure 3-7 shows the pseudocode of this algorithm.

The actual implementation of this algorithm uses two heuristics to help prune invalid options as soon as possible: (1) In step 3, the algorithm always picks the *largest* degree virtual vertex, so that the *Covered* set grows quickly to a set of highly connected nodes. This ensures that neighbors will be more likely to having many links to the *Covered* set and therefore less chances of having many valid mappings. (2) In step 5, while picking any neighbor would be correct, by choosing the one with more links to the *Covered* set forces the largest possible *conjunction* of constraints that must be satisfied, which helps prune invalid paths as soon as possible.

Due to its focus on exploring neighboring nodes and its preference for nodes that would result in less mappings to check for feasibility, this algorithm is referred to as the Lazy Neighborhood Search algorithm.

```

function LazyNeighborhoodSearch()
1) Covered, Neighbors  $\leftarrow \phi$ 
2) External sets  $\leftarrow$  set of all vertexes
3) Pick one vertex, move it to the Covered set
4) Vertexes connected to this move to the Neighbor set
5) current  $\leftarrow$  neighbor vertex
6) ConnectingEdges  $\leftarrow$  edges connecting current to covered vertexes
7) For all possible mappings for ConnectingEdges
8)   If this mapping satisfies all constraints
9)     Add current to Covered
10)    Update Neighbors
11)    If there are no more neighbors
12)      This is a good mapping
13)      Return to try alternative mappings
14)    Otherwise
15)      Go recursively to step 5
16)    Otherwise try another mapping for ConnetingEdges
    and if none passes, then return there is no mapping

```

Figure 3.7: The LNS Algorithm.

The implementation of LNS uses a data structure that holds in parallel the sets of covered, neighbor, and external vertices from both the query and hosting infrastructure, as well as boundary edges. It also keeps these sets synchronized when a virtual \rightarrow real node-pair moves in/out of the *Covered* set. The use of this data structure helps in several ways: (1) It facilitates the choice of the next neighbor (step 5) that has the largest number of links to the *Covered* set. If *Neighbors* is small, this selection will be more efficient. (2) It simplifies the identification of edges that connect the chosen neighbor to the *Covered* nodes by keeping the set of boundary edges.⁷ (3) It makes it easy to identify that a complete match has been found. Once the virtual neighbor set is empty, all the virtual nodes have been mapped and all

⁷Notice that this part is tricky since the correspondence of virtual boundary edges to real boundary edges is not one-to-one. This is why step 7 checks all possible mappings from a virtual neighbor to its real neighbors. For any positive match the algorithm recurses until all virtual nodes have been mapped, or no mapping is possible.

the constraints are satisfied.

Correctness of LNS:

Definition 1. *A promising mapping is a subset of any complete mapping. Observe that, if there is a mapping, the empty set is always a promising mapping.*

Lemma 2. *If, at the current state, the covered set of the LNS algorithm is a promising mapping of size k , then LNS will find a promising mapping of size $k + 1$.*

Proof. Let P_k be a promising mapping of size k at some stage of the LNS algorithm. Accordingly, the next neighbor selection heuristic selects a neighbor node n and tries all possible mappings r for this node. As P_k is promising, it follows that there must be at least one valid mapping for n and by trying them all. LNS will find that mapping and continue with the promising set of size P_{k+1} . \square

Theorem 1. *If there is any feasible mapping, the LNS algorithm will find it. If not, the algorithm will exit with empty solution.*

Proof. The first part of the theorem is proven by induction. The algorithm starts with the empty set which is promising. By lemma 1 it will find any promising mapping of size 1. At any state k it will find the promising mapping of size $k + 1$ and when $k + 1 = N_Q$ these are complete mappings. If there is no solution, the algorithm tries all extensions of the empty mapping, and after finding none, returns the empty set. \square

The correctness of the LNS algorithm follows from the fact that at any state P_k , LNS tries all possible mappings for the next neighbor and therefore it will find all feasible promising sets P_{k+1} , if any.

3.4 Network and Constraint Representation in NETEMBED

To be practical, the network embedding algorithms presented earlier require an easy interface with users and applications. For instance, such an interface should allow for an expressive specification of the topological as well as the qualitative characteristics of the hosting (real) and query (virtual) networks, and should also allow for

an expressive specification of constraints on acceptable embeddings. This section presents an overview of the choices made along these lines in the implementation of the NETEMBED service.

3.4.1 Network Representation

A network embedding service needs some “standard” representation of the various networks (graphs) that it handles, *e.g.*, query and hosting networks. To that end, there are a number of network topology representations available, but for the most part they have been designed for a specific application domain. For instance topology generators like BRITE [MLMB01] or GT-ITM [ZCB96] feature their own, different network description language. Simulation packages like ns-2 [NS209] are integrated with a programming language (TCL for ns-2) that provides the means to describe the network as well as to control the simulation. Experimental datasets also define their own means of characterizing the network. For example, in the particular case of the all-pair delay traces for PlanetLab [Yos06], the network is represented using an adjacency matrix that provides the minimum, average, and maximum delay measurements.

As varied as they are, the above-mentioned network representation approaches do not provide the generality necessary to describe the real and virtual networks with arbitrary parameters for nodes and links, as envisioned for NETEMBED. For this reason NETEMBED uses GraphML [BEH⁺01] as it provides a standardized way to describe the networks. In GraphML a network is represented as an XML document according to the rules of the corresponding *Schema Definition*, where the top-level element is the **graph** and its children are the **node** and **edge** elements. The advantage of this representation is that is simple and flexible. Moreover, integrity (syntax and type safety) come for free with the appropriate XML parser.

A particularly attractive feature of GraphML (as it relates to the applications

envisioned for NETEMBED) is its support of arbitrary *typed* attributes for nodes and links. For example a topology generation tool may give the average delay, while a measurement dataset may contain maximum and minimum delays. GraphML allows capturing all these attributes in the network description, and by means of the constraint expression language (discussed in the next section) the experimenter may establish the necessary relations between the corresponding parameters of the virtual and hosting networks.

3.4.2 Constraint Expression Language

The specification of a constraint expression (in addition to the topological constraints imposed on the links and nodes of the virtual network) allows for an independent input to the NETEMBED service and associated mapping algorithms, which enable users/applications the flexibility of specifying constraints that are separate from the query topology specification. This way, adjustments can be easily made without modifying the virtual network description. This may be useful in an interactive application scenario where the user may wish to begin with more stringent constraints and relax them if there is no compliant mapping.

To provide a general framework for specifying such relationships the NETEMBED implementation includes a constraint expression language, that basically follows the rules of Java for creating boolean expressions. The language provides the standard boolean operators (&&, ||, !), relational operators (==, !=, >, <, >=, <=), a basic set of arithmetic operators (+, -, *, /) and a few functions (*abs*, *sqrt*). It also follows the standard Java precedence rules.

The constraint expression is evaluated when comparing every edge of the virtual network with every edge of the hosting infrastructure. If such an evaluation returns a true value, the mapping between these edges is accepted. During each evaluation, the attributes of the links and nodes from both networks are available in the standard

Hosting Infrastructure	Virtual Network	Object
rEdge	vEdge	Edge object
rSource	vSource	Source node
rTarget	vTarget	Target node

Table 3.1: Objects available in NETEMBED constraint expressions.

dot notation under the names indicated in Table 3.1.

For example, consider a query posed against an overlay network such as PlanetLab, for which an all-pairs overlay link delay characterization is available. The virtual network may specify some requested delay values on its links, which must be matched up with overlay link delays. Furthermore, the query may specify some tolerated deviation around the requested link delays in the query network – *e.g.*, a 10% deviation around the requested delays is tolerable. The fragment below shows how such a constraint may be spelled out using the constraint expression language.

```
vEdge.avgDelay>=0.90*rEdge.avgDelay && vEdge.avgDelay<=1.10*rEdge.avgDelay
```

As another example, the fragment below specifies that a match is acceptable as long as the specified query link delay is within the minimum and maximum overlay network link delays.

```
vEdge.avgDelay>=rEdge.minDelay && vEdge.avgDelay<=rEdge.maxDelay
```

In some applications it may be necessary for some nodes in the query network to have special attributes that are not necessarily required of other nodes. To facilitate the expression of these relationships the constraint expressions may use the function `isBoundTo`. For example, if some query nodes have the attribute `osType` equals to `linux`, the following expression forces this nodes to be mapped to real nodes with the same attribute.

```
isBoundTo(vSource.osType, rSource.osType)
```

As another use example of the `isBoundTo` function, consider the case in which a particular binding needs to be enforced. For example, assume that a given query node must have access to special hardware (*e.g.*, a particular sensor), then the query may use the attribute `bindTo` to indicate this requirement, as shown in the fragment below.

```
isBoundTo(vSource.bindTo, rSource.name)
```

As another example of how the constraint expression language could be used to relate the specifications of a virtual network to the characteristics of the hosting infrastructure, the fragment below specifies the requirement that, in any valid embedding, the geographic distance between the desirable location of a query node and its corresponding hosting infrastructure node cannot exceed (say) 100 meters.

```
sqrt( (vSource.x-vTarget.x)*(vSource.x-vTarget.x) +
      (vSource.y-vTarget.y)*(vSource.y-vTarget.y) ) < 100.0
```

The implementation of NETEMBED uses the well known tools JFlex [JF1] and CUP [Hud] to implement the lexer and parser of the expression language.

3.5 Performance Evaluation

This section presents the results of an experimental evaluation of netEmbed, exploring its scalability under both synthetic and trace-driven workloads and infrastructures.

3.5.1 Experimental Setting

For each one of the experiments, and as illustrated in Figure 3-1, two “networks” need to be provided to NETEMBED: the query (or virtual) network, and the hosting (or real) infrastructure.

In the presentation that follows, the hosting infrastructures were either real infrastructures, namely PlanetLab [PACR03], or else they were generated synthetically using Internet topology generators, namely BRITE [MLMB01].

For the generation of the networks used as queries one of three approaches were adopted.

Using the first approach, the query network is a (typically small) subgraph selected at random from the hosting infrastructure. Such queries would be typical for applications that require the instantiation of Internet-like topologies, for example, as would be the case for configurations requested by a user of PlanetLab. One advantage of using this approach is that since the query is “sampled” from the hosting infrastructure, it is known in advanced that an embedding exists. This provides good “test cases” for NETEMBED.

Using the second approach, query networks are regular topologies that are synthetically generated (*e.g.*, rings, stars, cliques, *etc.*). Such queries would be typical for applications that exhibit a regular communication structure, as would be the case in high-performance grid applications, for example.

Using the third approach, query networks are irregular topologies that are synthetically generated using a topology generator such as BRITE. The motivation behind using this approach is similar to that of the first approach (since topology generators such as BRITE aim to generate “Internet-like” topologies). Additionally, using a synthetic generator allows studying the effect of the topological characteristics of the query network on NETEMBED’s performance, since it makes it possible to control such characteristics.

The main performance metric considered in these experiments is *response time*, which is the time (measured in milliseconds) it takes NETEMBED to answer a query. The response times reported for all experiments presented in this section were ob-

tained by running NETEMBED as the only active process on an Intel Xeon 2Ghz system with 1GB of main memory (enough for NETEMBED to avoid any noticeable paging).

3.5.2 Evaluation Using PlanetLab

These experiments use the PlanetLab all-pairs ping trace [Yos06] to generate the hosting infrastructure. This data set provides maximum, minimum and average delay between PlanetLab sites. Although there are 296 sites in the trace, some of the sites might not have been running the daemon or were down, so the actual number of active sites is a little lower and the underlying graph is not a clique. In any case, the network has 28,996 edges, providing a rich and large enough network for the tests.

The query networks for the queries were generated as random connected subgraphs from the hosting infrastructure (obtained as described above) of size N nodes. As alluded to before, setting the query to be a subgraph of the hosting infrastructure guarantees there is always at least one match. For each size query network size, queries were produced by varying the number of edges (E), and for each (N,E) -pair creating five different queries, so the results were not biased by a particular network configuration. The network embedding algorithms were run for each different query using the same constraint expression in all cases, namely that the real link delay range is within the specified query-link delay range.

Figure 3-8 (a) shows the performance results for the ECF algorithm. For each data point, the average and the 90% confidence interval are shown. A second line indicates the time to find the first match. This is an appropriate performance measure for applications that require just a single feasible embedding. Given the fixed size of the hosting infrastructure, the queries were limited to be up to 200 nodes in size. For the largest cases the running times were around 14 seconds on a Xeon

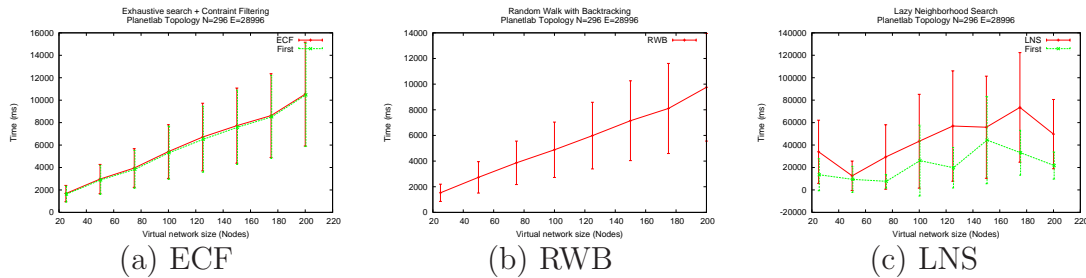


Figure 3-8: Mean Search Time for PlanetLab subgraphs

2.5Ghz system. It is worth noticing that, having a fixed-size hosting infrastructure, the search times seem to grow linearly with the size of the query, indicating that the filtering heuristic has been quite effective in avoiding the complexity associated with the full exploration of the search space.

An interesting observation from Figure 3-8 (a) is that the difference between the time to retrieve *all* matches and the time to find the *first* match is very small, indicating that most of the time was spent in the unmatched region of the search space – *i.e.*, once solutions are found, many similar solutions are found by varying just a few nodes (close to the leaves of the tree).

Figure 3-8 (b) shows the time to find the *first* solution using the RWB algorithm. Observe that in this case the mean search time shows a very linear relationship with the size of the query network.

Finally, the results for the LNS algorithm are shown in Figure 3-8 (c). Interestingly, LNS requires essentially constant time to find matches *independent* of the query network size. Even more interesting is the fact that for large query networks there is a slight tendency for the time to decrease. This behavior could be explained by noting that in large networks, there are usually very few matches, and growing a partial match implies matching a larger number of constraints.

Figure 3-9 provides a comparison of the three algorithms. Figure 3-9(a) shows the mean time until all matches are found, whereas Figure 3-9(b) shows the time

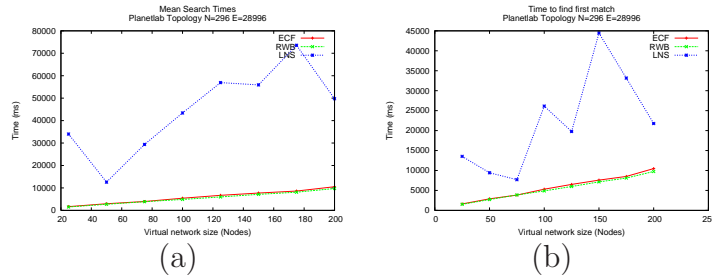


Figure 3-9: Comparison running queries with PlanetLab as hosting infrastructure

until the first match is found. The ECF and RWB algorithms have very similar performance, making it clear that the filtering strategy to prune the permutations tree is the prime contributor to the performance of these two algorithms, and that once a solution is found; most of the other solutions are close-by. On the other hand, it is interesting to observe that while on average the LNS algorithm is slow in comparison, when considering only the first match, its performance is not too far off. This could be explained by noting that LNS does not have the advantage of sorting the edges by number of potential matchings. Thus, when an edge that produces many matches appears early in the selection process, it will result in the repetition of a lot of redundant work for all the subsequent matches, which will result in significantly larger times if all matches are to be produced.

Another important question regarding the performance of the various algorithms is the time it takes them to conclude that an embedding is not feasible. Two sets of experiments were performed to evaluate this using PlanetLab as a hosting infrastructure. The first set of experiments uses queries that were known to be feasible (as was the case above). The second set of experiments uses queries that were known to be infeasible. The infeasible queries were generated from the feasible queries by changing some of their link attributes (*e.g.*, delays) to some infeasible values. Notice that doing so does not change the topology of the query network, only the constraints imposed on what would constitute a feasible embedding. Figure 3-10 shows the results

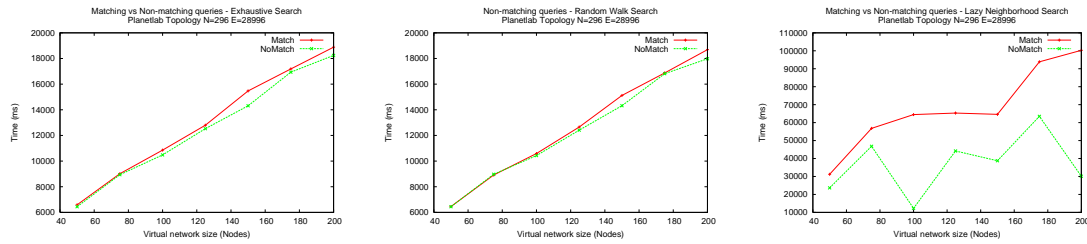


Figure 3-10: Search times for feasible versus infeasible queries

for the three algorithms. In general, the performance of ECF and RWB are very similar, which is consistent with the fact that these algorithms will explore a similar fraction of the search tree in either case. LNS is noticeably slower. In the cases where there are not feasible matches LNS returns the no-match result significantly faster.

3.5.3 Evaluation Using Synthetic BRITE Topologies

Exploring the scalability of these algorithms is not possible with PlanetLab’s topology given its fixed size. Also, since the delays on PlanetLab links are characterized using end-to-end measurements (pings), the resulting topology is mostly a clique, which is far from being representative of physical (as opposed to overlay) Internet topologies. For these reasons, synthetic topologies generated using the BRITE topology generator were used for exploring the scalability of these algorithms.

These experiments were conducted with 3 hosting networks of size 1500, 2000 and 2500 nodes respectively. For each one of these networks, various sets of sub-networks of different sizes were obtained. The results of the three algorithms for these networks are shown in Figure 3-11. In general, and not surprisingly, the behavior follows the same pattern observed with PlanetLab topologies: similar average times for ECF and RWB, with a slight advantage to the latter; high variability and much larger mean times for LNS. Figure 3-12 shows the mean time to find the first match for these BRITE cases. Again, the difference between ECF/RWB and LNS is not so

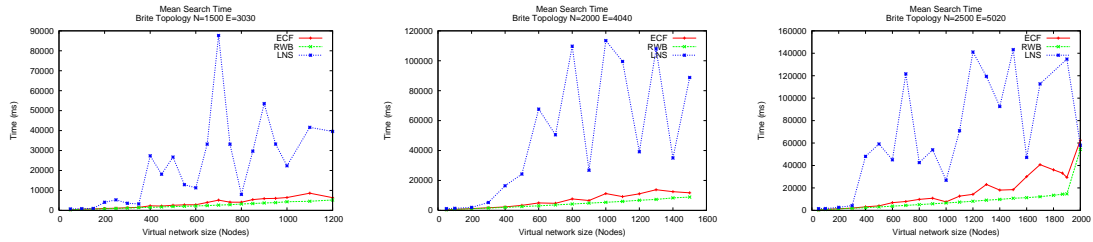


Figure 3-11: Mean Search time for BRITE topologies

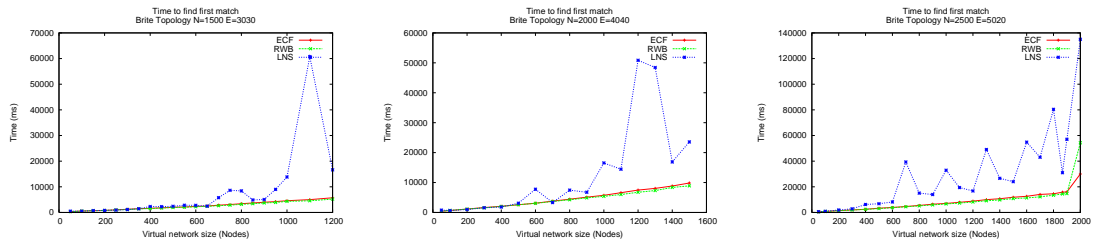


Figure 3-12: Mean Time to Find First Match in BRITE topologies

pronounced.

3.5.4 Evaluation Using Queries with Regular Topologies

The two characteristics that make an embedding difficult to find are: (1) under-constrained queries, and (2) queries with regular topologies. Under-constrained queries do not provide enough conditions to significantly prune the search space. In the limit, the only constraint is that of the query topology, and the problem is reduced to a subgraph isomorphism problem. With regular topologies (such as cliques, rings, stars with equal or no constraints on all edges), any permutation of a partial match is also a partial match. Thus, if this partial match leads to a dead end, the embedding algorithms will end up performing the same amount of (useless) work on every permutation.

To evaluate the performance of NETEMBED’s algorithms under these “worst-case” scenarios, a series of cliques of increasing size were used as queries. The only constraint was to have a end-to-end delay between 10 and 100ms. Then each one of

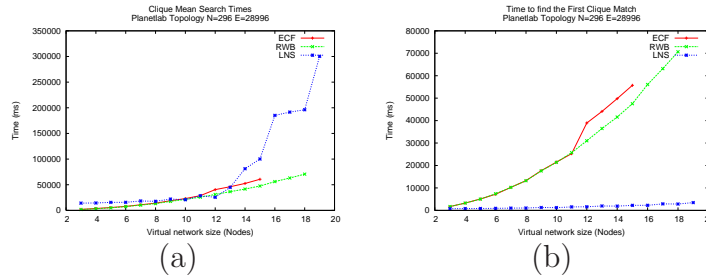


Figure 3-13: Finding Matchings for a Clique

these queries is run against a PlanetLab hosting infrastructure. The query is under-constrained as there are about 6,700 edges that fall in this delay range and the query topology is regular.

Figure 3-13 (a) shows the mean time to find *all* embeddings of a clique in PlanetLab as a function of the clique size. Those cases in which no solutions were found, or in which the algorithm timed-out before returning any solution are excluded, so as not to affect the trend of the graph. It is worth noting that under this query model, LNS always times out before it is able to find *all* embeddings.

Figure 3-13 (b) compares the three algorithms using the time to find the first match. In this case, the LNS algorithm greatly outperforms the other two. When it finds a solution it finds it quickly as the heuristic to grow the matching with the vertex with more constraints helps prune non-matching cases rapidly as this forces each new vertex to match all the already selected vertexes. On the other hand, the regular structure forces the algorithm to start over when finished with a candidate matching explaining the long running times for the average case.

A last set of experiments were conducted involving composite queries. A composite query is a two-level hierarchical topology, where both levels have regular structures. So for example the root level could be a ring, a star, or a clique, and each vertex of the root level is also a regular structure. Many practical applications follow these kinds of structures, including multicast trees, distributed hash tables, to

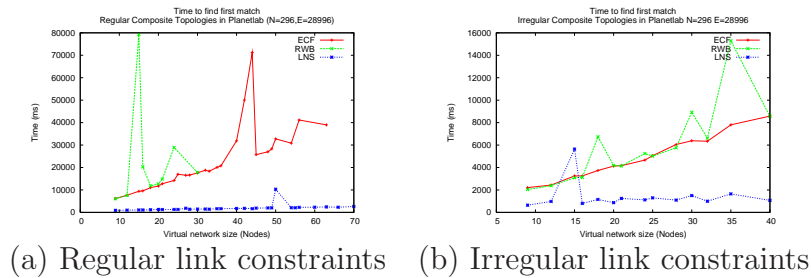


Figure 3-14: Finding matchings for composites of regular topologies

name some examples. Two sets of queries were prepared. The first set has regular constraints per level – namely, all links at the root level have a delay constraint between 75 and 350ms, representing inter-site wide-area delays, and all the links on the lower level have delay constraints between 1 and 75ms, representing intra-site delays. These values of the delays were chosen from the distribution of delays in PlanetLab so that there are abundant links in both ranges. The second set shares the same topological structures as with the first set, but features delay constraints that are randomly assigned from the 25-175ms range, which contains about 70% of the links in PlanetLab. For both sets, there are usually thousands of matchings for each query, so the interesting measure here is the average time to find the first match. Figure 3-14 shows the results for (a) the regular and (b) the random constraint assignment cases.

The interesting observation about these two cases is that (as with the first match in the case of cliques) LNS finds the first solution in almost constant time and by far outperforms the other two algorithms. This reinforces the previous conclusion that in under-constrained queries and high-density graphs LNS is better suited to find the first solution.

3.5.5 Quality of Returned Results

Using any one of the embedding algorithms, NETEMBED may return one of three types of results: (1) The complete set of all feasible embeddings, (2) A subset of all feasible embeddings, and (3) An inconclusive response.

The complete set of all feasible embeddings (including none, if the query network is impossible to embed) is returned when the algorithm terminates before its preset timeout has expired. A partial set (subset) of all feasible embeddings is returned when the algorithm times out after finding some (but not necessarily all) feasible embeddings.⁸ Finally, the algorithm is said to have an inconclusive response, if it fails to produce any feasible embedding by the timeout. Notice that in this case, it is inconclusive whether or not a single feasible embedding exists.

Figure 3-15 shows the probability of each one of these results for all the experiments presented earlier in this section. Except for one case, in which RWB behaves poorly, the probability of finding matches was over 70%. Even more impressive, for some types of queries, ECF and LNS were able to find all feasible matches with a probability of 75% to 82%. Looking at the probability of finding any embedding (as opposed to all embeddings) for ECF and LNS, observe that for queries with regular topologies (clique and composite), LNS has a better chance of success. This, combined with the much better performance in terms of response time, makes LNS ideal for these kinds of queries. On the other hand, for very constrained queries, where filtering results in much more effective pruning, ECF outperforms LNS in both chances of success and response time.

⁸Notice that RWB will always return a partial result (or no results) since by design it terminates as soon as it finds the *first* solution.

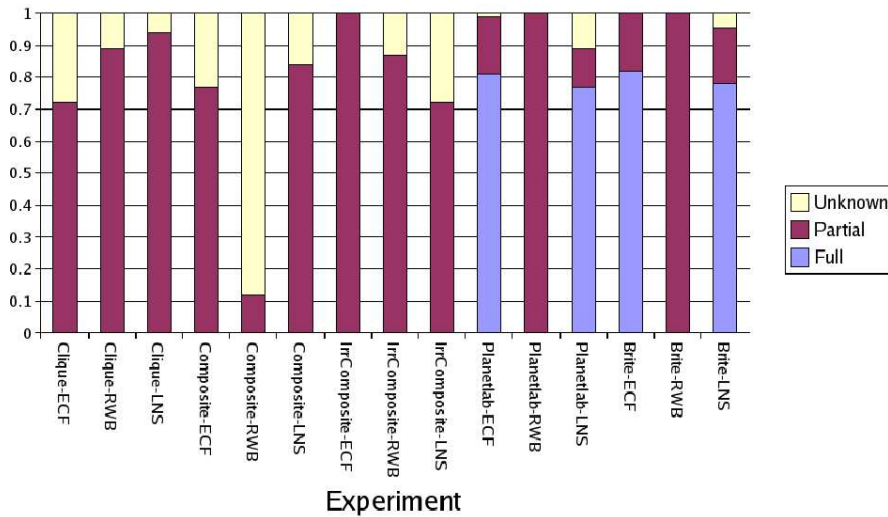


Figure 3-15: Probability distribution of the different types of results

3.5.6 Performance Results Relative to Existing Techniques

Previously published techniques ([ALR03], [WLS⁺02], [WLG02], [ZA06], [CBMp03], [OAPV05]) give a baseline for the purpose of comparison. Most of these existing techniques handle only small networks, typically tens of nodes, and have reported response times in the order of tens of minutes or more. As it has been shown in the previous sections `NETEMBED` handles much larger networks with response times that vary from a fraction of a second to a few minutes depending on the query ⁹

3.6 Related Work

The network embedding problem, as an optimization problem, appears in many distributed application environments. The following is a synopsis of the most representative works classified according to the application area.

⁹It is hard to make a direct comparison between these various techniques since, as alluded in Chapter 2, most of these techniques are designed to find optimal mappings and/or are designed to deal with very particular kinds of mappings.

3.6.1 Resource Discovery and Allocation for Testbed Platforms

Testbeds platforms such as Emulab [WLS⁺02], PlanetLab [PACR03], and GENI [GEN10] offer resources for running large-scale experiments on networking and distributed systems. The first two have been operational for many years now, therefore the research community has had the opportunity to experiment with various resource management schemes. In the case of GENI, which as of 2010 is still under development, its underlying resource management mechanisms have not yet been published.

Emulab's core is a centralized pool of servers interconnected by high-speed Ethernet LANs. The servers themselves belong to a few classes of homogeneous servers. When instantiating an experiment, some of the servers act as nodes for the application, while other servers perform the function of emulating network links, for which transmission rate, packet losses and propagation delay may be programmatically assigned. In addition to the emulated resources, an experiment can combine simulated resources (via NSE an extension for the NS-2 simulation package [NS209]) and real resources from PlanetLab. This gives the experimenter a large amount of flexibility when instantiating experiments. Resource management in Emulab is centrally controlled and the goal of the allocation mechanism is to maximize the number of concurrent experiments. The problem has been identified as an NP-hard problem; therefore various heuristics have been proposed to find feasible solutions: *wassign* [WLS⁺02] uses a genetic algorithm and *assign* [ALR03] uses a simulated annealing algorithm. In both cases, a scoring system is used to measure the quality of a solution and the solution that places the least demand on the more limited resources (*e.g.* inter-switch bandwidth) is preferred.

PlanetLab is a distributed testbed whose resources are provided by academic and industry institutions. An institution provides some servers with Internet con-

nectivity, and relinquishes the control of these servers to PlanetLab Central, which administers all PlanetLab nodes. The contributed servers represent a heterogeneous mixture of machines from different vendors and hardware generations. For instantiating an experiment the user submits a request specifying the set of nodes, and if accepted, PlanetLab central creates a *slice* which embodies the set of resources (nodes) allocated to the experiment. The *slice* is composed of one *sliver* per allocated node, and the *slivers* are instantiated on the servers using an extension of the Linux Virtual Server technology which provides a limited form of resource isolation between colocated *slivers*.

As opposed to Emulab where the topology is instantiated to match query specification, in PlanetLab the underlying infrastructure is fixed and the experimenter's goal is to find a set of nodes that match the query specification. For this reason SWORD [OAPV05, AOPV08], a mapping service for PlanetLab, was designed to minimize the *mapping error*: an indicator of how close is a given set of nodes from a query specification. To do this, SWORD maintains a database of static and dynamic information about PlanetLab nodes and their links. Dynamic information such as average CPU load, available bandwidth and inter-site latency is periodically updated by using several monitoring services. SWORD query language adopts a simplified topology representation, supported by the fact that it captures the needs of typical applications. In this model, the requested resources are divided in groups, such that the requirements for all the nodes within a group are very similar. The query language permits the specification of *intra-group* requirements and *inter-group* requirements. Intra-group requirements constrain the choice of nodes for the group based on both, per-node and node-connectivity requirements. Inter-group requirements specify the connectivity between groups in terms of available bandwidth and latency. The queries also include *penalty functions* which give a notion of how good

a given mapping is. Since the mapping problem is an NP-hard problem, SWORD adopts a search technique that leverages the hierarchical structure of the query language. In a first phase, it evaluates the per-node penalties and enumerates feasible node candidates for each one of the groups. In the second phase, it computes the intra-group penalties among the possible node-candidates and again returns the group candidates with lowest penalty. In the last phase, it uses the candidate groups from the previous phase to evaluate inter-group penalties and returns those mappings that give a small total penalty. By adjusting the penalty threshold and the time-outs for each one of the stages it is possible to bound the run time of the matching algorithm. The main tradeoff of SWORD is that it is susceptible *false negatives*: a not found answer, even when solutions might exist.

3.6.2 Overlay Networks

Overlay networks have been amply considered in the literature as the means to improve performance metrics observed by applications. So, for example, Resilient Overlay Networks (RONs) [ABKM01] aim to improve the reliability, while Service Overlay Networks (SONs) [DZH03] aim to provide QoS guarantees in the form of bandwidth and end-to-end delays.

In the case of RON, the overlay nodes operate as a distributed system where all nodes *cooperate* in implementing the routing function. Although RON does not consider the possibility of selfish nodes joining the overlay, it could occur in practice. A selfish node could easily *free-ride* the service by “lying” on his link-state advertisements¹⁰.

In the case of SON, overlay nodes are assumed to be under a single ownership, therefore the overlay owner enforces node cooperation. The topology of the overlay

¹⁰When node A sends a link state message to any of its neighbors, say node B, it only announces knowledge of the link A-B. So, no neighbor will find routes going through A.

and the routes assigned to each flow are inputs to the system. In order for the overlay to be able to provide a guaranteed service, it acquires the capacity on each one of the underlay's links under the terms of a Service Level Agreement (SLA). This process establishes the costs associated with each underlying link. Given the costs of the resources and the revenue made by the overlay, the problem for the owner is a utility maximization problem subject to QoS constraints, whose approximate solution can be computed analytically or numerically.

When the set of routes is not known/given in advance, overlay creation becomes a more challenging problem. The works by Zhu and Ammar [ZA06] and Yu *et al* [YYRC08] are example works that consider this situation. In the first case, the assumption is that a global entity (say, the system operator) has control over all the overlays that need to be created. The operator's problem is to assign underlay resources (nodes and links) to all the overlays in such a way that it minimizes a combination of the maximum link and node *stress*. The *stress* is a utilization metric, so for the case of overlay nodes, the stress is proportional to the number of users sharing the node, and for network links, it is proportional to the total capacity allocated on the link. The motivation for this metric comes from the fact that minimizing the maximum stress helps avoid saturating bottleneck resources, giving more chances for being able to allocate new overlays when they arrive, and minimizing the impact on the performance as, by standard queueing models, the service delay is heavily dependent on resource utilization.

The work by Yu *et al* [YYRC08] also considers the case where multiple overlays need to get resources from a common substrate infrastructure. In this case, associated with each overlay is a revenue function which measures the benefit obtained by the overlay customer as function of the virtual nodes and links that compose the overlay over time. The goal of the system is then to find mappings that maximize the

total revenue. The solution approach uses greedy heuristics (as finding the optimal is an NP-Hard problem). These heuristics split the mapping process in two phases: Mapping the virtual nodes, and then mapping the virtual links. The first phase sorts the request in decreasing order by their revenue (noticing that the revenue is independent of the chosen mapping) and allocates nodes for them for as long as there is capacity left in the substrate nodes. The second phase allocates virtual links using the k-shortest path algorithm to search for a path that satisfies the bandwidth requirement of the overlay link. They also present an alternative method for mapping virtual links: As the revenue per virtual link is independent of the mapping to the substrate, virtual links can be split into fractional flows over various substrate paths. By doing so, the virtual link mapping problem can be transformed into a fractional multicommodity flow problem, which is solvable in polynomial time using standard linear programming algorithms. Being a heuristic approach, there are no assurances on the optimality of the solution and it is also prone to return false negatives.

3.6.3 Service Oriented Architecture (SOA)

Many large scale development systems use the notion of composable services, (*e.g.* SOA, EJB, DCOM, CORBA, RPC, Grid/Globus, MPI). A high-level application functionality is built by composing some of those services. The resulting application is typically represented by a Directed Acyclic Graph (DAG). The actual services may be assigned to the substrate resources in many possible ways, but the performance of the application is heavily dependent on the way such assignment is made. For example if two components transfer large amounts of data between them, is better to place them closer together to avoid the large latency of delivering the data over low capacity links.

An example architecture for allocating resources in a service composition environment is presented by Yu *et al* [YL05, YZL07]. In this system the applications are

represented as DAGs and have quality of service attributes which must be satisfied by the assigned substrate resources. A utility function is defined in terms of the quality parameters, so that larger utility corresponds to an assignment that gives better application performance. When the DAG has a sequential flow structure, the allocation problem reduces to a Multidimensional Multichoice Knapsack Problem (MMKP). In the case of general graph structure the problem reduces to a Multiconstrained Optimal Path (MCOP) problem. Since both problems are NP-complete, the authors propose several heuristics to handle each case. In their work, the utility function is defined per application, and a central broker executes the search algorithms to find feasible locally-optimal solutions. The implicit assumption in this model is that allocations are enforced by the system and that different applications do not interfere with each other, therefore their utilities are independent (the actions of one user have no effect on others).

Another representative example is SpiderNet [GNY04]. Similarly to the previous example, applications are represented as DAGs and have predetermined QoS goals. The main difference is that the SpiderNet architecture is fully distributed. To this end, it defines a protocol for discovering feasible mappings that all participating nodes are assumed to follow. It also offers the capability of finding backup mappings for quickly recovering from failures. As there may be multiple mappings for any given application, and many applications may share the same set of basic services, the allocation algorithm uses as an objective of the optimization a metric defined as the weighted sum of demand-to-availability ratios over all the resources. The mapping for an application is chosen as the one that minimizes this metric, motivated by the fact that doing so helps load-balance the resources. The discovery protocol uses a limited-flooding strategy for the discovery process, which helps improve the scalability of the system. This heuristic makes the system susceptible to false negatives, as a feasible

solution (or the optimal one) may be pruned in the limited flooding phase.

3.7 Summary

Large distributed testbeds are the backbone of new applications, some of which will possibly reach maturity and be deployed on the Internet. Service providers aware of the market opportunity are already offering dynamic on-demand infrastructures where these services can be deployed in various forms of Cloud Computing offerings. The problem of selecting the resources over which a given system is going to be deployed permeates from the testbed to the Cloud environments and the demands on scalability and performance are expected to be more stringent as these systems are used for commercial purposes.

NETEMBED addresses this problem by exploiting the fact that embedding problems include various forms of constraints, which help prune and efficiently explore the search space, making the problem tractable for instances of practical size.

As for the search problem itself, NETEMBED provides various heuristics to find the one or more feasible mappings for the query application. These heuristics are proven to be free of false positives and false negatives, although occasionally may be computationally intensive.

The scalability of NETEMBED was studied on an actual implementation of the service, using a set of representative infrastructures and queries that capture some of the most relevant properties needed for real applications, showing its ability to handle problem instances of practical sizes.

Chapter 4

Colocation Games: Sharing Resources by Splitting the Costs

The previous chapter presented NETEMBED as a framework for applications that need to find resources from a distributed infrastructure. Using NETEMBED the problem was limited to the decision of a single application. An immediate extension is to consider the case of multiple applications sharing the infrastructure. A trivial case is that where the utility/cost for each application is independent, therefore the best mapping depends only on the current system state (basically what resources are or are not available) and not on how the resources are shared. When the way of sharing the resources affects the utility/cost for each user, the actions of the users are no longer independent. This scenario is typically modeled as a strategic game under the rationality assumption:¹ each user will do what is best according to its own utility function. This chapter explores the network embedding problem under the assumption that rational users share the infrastructure.

The relevance of this problem is supported by the many examples of distributed computing infrastructures, designed to be shared among many customers, *e.g.* Infrastructure as a Service (IaaS) offerings, testbed platforms, network overlays, P2P systems, grid computing, *etc.* As discussed in Chapter 2, it is the case in many of

¹It should be noted that for the purposes of this work in general, “the user” is understood as a software agent acting on its behalf. The rationality assumption itself has been seriously questioned [Fol07] in economics, but this work assumes that the algorithmic agents interacting in these systems are impervious to other motivations.

these systems that users can make their own allocation decisions. In doing so it is expected that users will follow a strategic behavior whereby their actions reflect their intent to improve their own utility/cost. The existence of federated computing infrastructures and technologies such as virtual machine and network virtualization makes this strategic behavior possible: First, because there is no single authority controlling the system, and second because switching from one configuration to another becomes easier and relatively inexpensive.

Existing literature on algorithmic game theory has explored various forms of games where sharing resources makes the utility of users interdependent. For example earlier works such as [KP99, CV02] explore the problem of routing n flows through m parallel links (or equivalently executing n tasks in m parallel processors), under the assumption that flows (tasks) can be fractionally split. All flows (tasks) share a fraction of the capacity of each link (processor) and the time it takes to complete a job increases with the number of flows sharing the link. Therefore, the strategy for each flow is to split its work among the m links in such a way that minimizes its completion time. This system is modeled as a *mixed-strategies game*. It is shown that a Nash Equilibrium (NE) always exists and that the Price of Anarchy (PoA) is bounded. These works have been extended to consider the cases where the network has an arbitrary topology [RT02], or the flows/tasks cannot be fractionally split [FKK⁺02, AAE05]. The later cases are modeled as a *pure-strategies game*.

The previous examples focus on the performance as the metric that defines the utility for the user. Rational actions are those that improve the performance for the respective user. Observe however that for scenarios where there is a reservation system in place, performance is no longer a motivator of user actions. Each user is guaranteed a minimum QoS by means of a SLA with the service provider.² Viola-

²Although not widely available yet, it is expected that as customers demand it, providers will

tion of the SLA incurs on a penalty for the provider. On the other hand, there is an explicit pricing scheme and it is the user’s goal to minimize its own cost. Also, it is common for providers to quantize the allocation units, for example offering small/medium/large instances of its resources. Under these circumstances, it is possible for users to benefit from sharing some of these resources. For example, if users A and B need an instance larger than the small instance, but they would underutilize the medium instance, it may be the case that by sharing a large instance they both get what they need and end-up paying less than the cost of two medium instances. It is possible then for users to engage in a strategic behavior with the purpose of improving their own positions. In particular, the *Colocation Games* introduced in this chapter model this strategic behavior for the case when the users split the cost of the resources in a fair manner: *each user pays in proportion to the fraction of the utilization due to his applications.*

Fairly sharing the cost of the resources has been previously explored in the case of routing flows in a network [CR06]. In this case each flow has an associated weight and the payment to each link is the Shapley share of the sum of the weights sharing the link. As a matter of fact, this model is very close to the *Colocation Game*, except that the *Colocation Game* generalizes for arbitrary subgraphs (instead of source-destination flows), allows for different weights on each link and each node (instead of having a constant flow from source to destination), and also incorporates the notion of constraints (such as performance or functionality constraints – as previously covered in Chapter 3).

This chapter begins by extending NETEMBED’s model (§4.1) in order to incorporate the sharing of resources by multiple users³ using the *Shapley cost* as the mechanism for fairly splitting the costs. Section 4.2 presents the analytical results,

begin offering guaranteed services spelled out through SLAs.

³This text uses the terms ‘users’ and ‘applications’ interchangeably to indicate each one of the independent entities interested in allocating some resources.

which include proving that the decision problem of determining the existence of a NE is NP-complete and provides several special cases in which the NE always exists. For some of these cases it is also proven that for the *Colocation Game* better/best-response dynamics always converge to the NE. These cases are interesting as they provide the means for implementing a fully distributed, online and self-organizing resource management system, such that the selfish/rational actions of the users lead to a mapping of resources that achieves high utilization. The sketch of this implementation, including various heuristic solutions to the better response problem, is presented in Section 4.3. Finally, Section 4.4 presents a performance evaluation of the Process Colocation Game (PCG) using both, synthetic and real traces. These experiments corroborate the analytical results and also highlight the actual efficiency of the PCG as a mechanism for allocating resources in a distributed environment. The experimental results indicate that in practice the overall utilization obtained by this mechanism is much better than the worst-case bounds established during the analysis.

4.1 Colocation Games: Definitions

Back to the context of the network embedding problem (§3.2), assume there is a hosting infrastructure $G = \langle V, E \rangle$. Vertices (or nodes) in G represent standalone resources, whereas edges represent relationships between these resources. Examples of standalone resources include processors and storage, both of which would be represented as vertices in G . Examples of relationships between standalone resources include communication links and dependency relationships, both of which would be represented as edges in G . Note here that edges in G may be directed or undirected. For instance, a pipeline of processing units would be represented using a chain of vertices, in which each pair of adjacent vertices are connected using a directed edge,

whereas a cluster of processing units on a LAN would be represented using a fully-connected subgraph, in which each pair of vertices are connected using an undirected edge.

Let the labeled graph T model the set of resources and underlying relationships that are necessary to support a specific user application or *task*, also called a query in G . This graph will be referred to as the user requested task graph. Vertices and edges in T have the same meaning as those in the hosting graph G . Labels in G or T may decorate either vertices or edges. In a hosting graph, labels specify *supply* attributes such as unit capacities and unit prices of processing or communication links.⁴ In a task graph, labels specify *demand* attributes such as the minimum CPU utilization and storage needed by a standalone process, or the minimum bandwidth tolerable by communicating processes, *etc.* in the task. Notice that resources may have *multidimensional capacities*. For example, a Virtual Machine (VM) instance specifies capacities along three dimensions corresponding to CPU, RAM, and disk. Thus, it is convenient to refer to the supply of (or demand on) a resource with a capacity (utilization) vector.

As illustrated in Figure 4-1, a set of requested task graphs (one per user) constitutes the overall *workload* to be hosted on the infrastructure graph G . A *mapping* M of the (vertices and edges in the) set of request graphs to the (vertices and edges in the) hosting graph constitutes a configuration underscoring a specific assignment of users to resources. A *valid configuration* is one wherein supply meets demand – for example, the aggregate demand (*e.g.*, CPU utilization) of all users sharing a vertex in G do not exceed the supply (*e.g.*, CPU capacity) of that vertex. Given a valid configuration, the infrastructure provider expects to be paid for any resource in G

⁴Without loss of generality, this chapter uses capacity and pricing, noting that other attributes may well be considered, *e.g.*, the maximum delay on a link or the OS version and supported APIs on a processing unit. Such attributes would be used to determine the feasibility of mapping vertices and edges from T to G , same as it was the case in Chapter 3

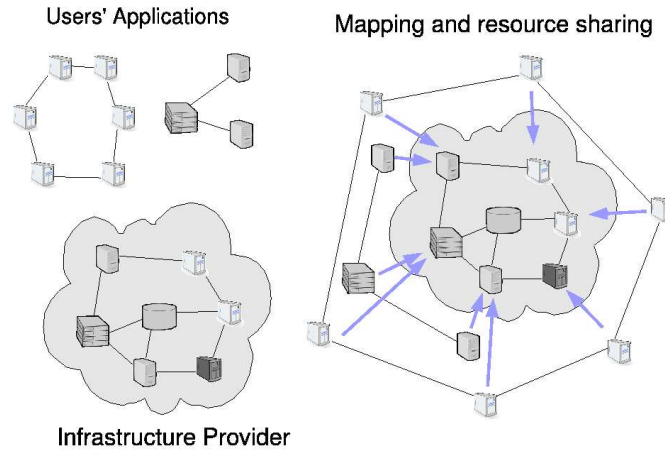


Figure 4-1: The Colocation Game

used by at least one task (user), but not for (idle) resources to which no tasks were mapped. The price charged per resource is fixed, independent of the number of users sharing that resource. The cost incurred by a user is given by a *cost function*, which apportions the price of each resource in G among all users with tasks mapped to that resource. The cost function can be seen as the marketplace mechanism that governs and induces symbiotic relationships among rational, selfish agents (the users). This thesis adopts a specific form of cost functions that may be conceived as fair – namely, those that split the fixed cost of a resource among tasks in some proportional (*e.g.*, linear) fashion based on the utilization of that resource by the various tasks assigned to it. This cost function is known as the *Shapley Cost* [NRTV07].

4.1.1 The General Colocation Game (GCG):

Given a hosting graph $G = \langle V, E \rangle$ where each vertex and edge in G is labeled with a *resource capacity vector* (R) and a *price* (P), and given a collection of tasks, each in the form of a graph $T_i = \langle V_i, E_i \rangle$, where each vertex and edge in T_i is labeled with a *weight* underlying a resource demand vector (W), the *General Colocation Game* is the pure-strategies game, in which each task is able to make a (better response)

move whereby, if possible, the task modifies a valid mapping M into another M' so as to minimize its own cost, given by a function $c_M(T_i)$ for the cost of task T_i when hosted in G according to a mapping M :⁵

$$c_M(T_i) = \sum_{j \in \{V_i, E_i\}} P_j \cdot \left(w_{ij} + (1 - U_j) \frac{w_{ij}}{U_j} \right) = \sum_{j \in \{V_i, E_i\}} P_j \frac{w_{ij}}{U_j} \quad (4.1)$$

where w_{ij} is the weight (or utilization) imposed on resource j by task T_i , P_j is the price of the resource, and U_j is the total utilization of the resource (by all tasks assigned to the resource by M). This cost function assures the budget is balanced: the sum of payments to any individual resource is equal to its price.

The *social cost* of GCG for a given mapping M is the sum of the costs of all tasks:

$$s = \sum_{\forall T_i} c_M(T_i) \quad (4.2)$$

4.1.2 The Process Colocation Game (PCG):

PCG is a restricted (simpler) version of GCG. This simplified version is interesting as it possesses some properties not present in the GCG. In a PCG, a task graph consists of a *single vertex* representing an independent process (say a computation) that needs to be assigned to a single resource (say a processor). In a PCG, the cost function for process i when mapped to resource j is

$$c_j(i) = P_j \cdot \frac{w_i}{U_j} \quad (4.3)$$

where U_j is the overall utilization of resource j , which must satisfy its capacity constraint:

$$U_j = \sum_{i \in j} w_i \leq R_j \quad (4.4)$$

⁵Throughout this chapter, the processes and resources of an user are referred to by the user's index, *e.g.*, i instead of T_i .

Assuming all processes (users) are rational and selfish, the only move that a process would make in PCG is one that results in a reduction of its own cost *and* (as will be shown later) would also benefit other processes with which the process would be colocated as a result of the move. When a process i moves from resource a to resource b , two situations are possible. In the first situation, the move by process i does not result in a displacement of any of the processes on resource b . These are called *placement moves*. This situation occurs if resource b has enough capacity to host process i (in addition to all the processes already on it before the move). In this case, it is easy to show that the utilization of resource b increases and that the cost for all processes already on resource b is reduced. In the second situation, the move by process i results in displacing one or more of the processes that were on resource b before the move. These are called *replacement moves*. For a replacement move to be possible the cost of all processes that are not replaced by the move *must* be reduced (another way to look at this move is to think of the final set of players as a *coalition* – they join together in b as this reduces the cost of all the members of the coalition). This will be the case if the move results in a strict increase in the total utilization of b .

Let U'_b refer to the utilization of resource b after a move by process i into it. Clearly, for both placement and replacement moves, $U'_b > U_b$ – *i.e.*, U'_b increases as a result of the move. The only reason for a process i to “move” from resource a to resource b is that $c_b(i) < c_a(i)$. This yields the following conditions for what constitutes a *valid move* by process i into resource b :

$$U'_b > U_b \quad \text{and} \quad \frac{P_b}{U'_b} < \frac{P_a}{U_a} \quad (4.5)$$

4.1.3 The Multidimensional Process Colocation Game (MPCG):

As alluded to before, a resource (vertex) in the hosting graph may have a multi-dimensional capacity (recall the CPU, memory, and storage attributes of an EC-2 instance) and is thus represented by a capacity vector $R_j = [r_{j1}, \dots, r_{jd}]$ and a fixed (scalar) price P_j . A natural representation of the demand from such a resource is a vector $X_i = [x_{i1}, \dots, x_{id}]$, whose components represent the task's demand (or requested utilization) from each dimension of the resource. A MPCG is a game in which resources (and process utilizations) are multidimensional.

For the purpose of defining a cost function and valid moves for MPCG, the definitions from the previous section need a few adjustments. In particular, a set of tasks can be assigned to resource j if the sum vector of demands is component-wise less than the capacity vector:

$$\sum_{i \in j} X_i \leq R_j \quad (4.6)$$

To apportion the multidimensional resource price among the processes collocated at the resource, define the multidimensional utilization of a single process by its volume: $v_i = \prod_{k=1}^d x_{ik}$. The total utilization of all processes collocated at resource j is the sum of the multidimensional utilization of these processes: $U_j = \sum_{i \in j} v_i$. This leads to the following definition for the cost of task i when allocated to resource j (noting again that the sum of the cost for all tasks collocated at resource j matches the price set for resource j):

$$c_j(i) = P_j \cdot \frac{v_i}{U_j} \quad (4.7)$$

In a MPCG, a *valid move* is one that meets the feasibility constraint (4.6) and that gives a cost reduction for the task, so when process i moves from a to b , $c_b(i) < c_a(i)$ and $\frac{P_b}{U_b+v_i} < \frac{P_a}{U_a}$. As before, this definition extends to *replacement moves* as well, by assuring that the task that moves and the tasks that remain in the destination

reduce their costs

$$U'_b > U_b \quad \text{and} \quad \frac{P_b}{U'_b} < \frac{P_a}{U_a} \quad (4.8)$$

4.1.4 The Parallel Process Colocation Game (PPCG):

This is an extension of PCG which allows for tasks to consist of a set of parallel processes. In PPCG, the hosting graph G is a complete graph of n vertices (or nodes), where the i^{th} node has resource capacity R_i and price P_i . In PPCG, a workload consists of a collection of task graphs T_1, \dots, T_m , where task graph T_i consists of a set of k_i nodes, each of which with a specific utilization requirement (weight). Thus T_i is fully specified by parameters $(k_i, w_{i1}, \dots, w_{ik_i})$, where k_i is a positive integer denoting the number of nodes (*e.g.*, processors) needed by T_i and $0 < w_{ij} \leq 1$ denotes the utilization demanded from node j .⁶ Notice that by definition, the k_i nodes requested by T_i must be mapped to different nodes in the hosting graph G . When $w_{i1} = w_{i2} = \dots = w_{ik_i}$, in this case T_i is called a *uniform parallel process* and the resulting game is referred to as a *uniform PPCG* (non-uniform PPCG, otherwise).

As described above, in PPCG there are no topological constraints on the set of nodes requested by a task (process) since graph T_i featured no edges. In general, however, a parallel process may specify a particular topology, which must be satisfied in any valid mapping of the task graph onto the hosting graph. Of particular interest are parallel process colocation games in which the parallel processes have regular topologies (*e.g.*, a ring, a mesh, a hypercube, *etc.*) Also, as described above, in PPCG the hosting graph was a complete graph. In general, however, the hosting graph itself may feature a regular topology onto which the parallel process structures would be mapped. Again, of special interest are hosting graphs that exhibit regular topologies.

⁶Clearly, standard PCG is a special case of PPCG where $k_i = 1$.

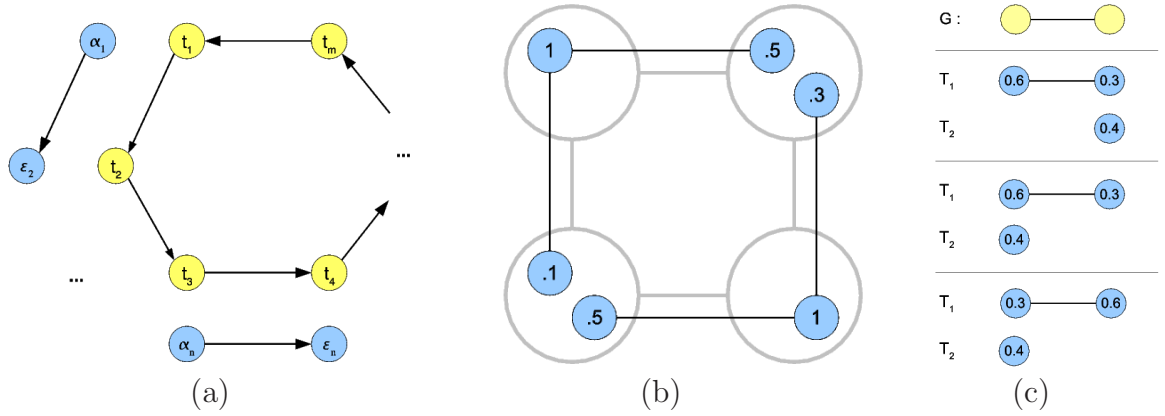


Figure 4-2: Examples of Colocation Games with no Nash Equilibrium.

4.2 Analytical Results

Nash Equilibrium of GCG: The GCG does not necessarily have a NE, as illustrated by the examples in Figure 4-2.

Consider example (a), where the hosting graph is an m -vertex ring, and each vertex is of unit capacity. Each of the n ($2 \leq n < m$) tasks consists of two connected vertices, with utilization requirements $1/2 < \alpha_i \leq \alpha_{max} < 1$ and $0 < \epsilon_i < 1 - \alpha_{max}$, respectively. Observe that two tasks, say T_1, T_2 , cannot be assigned to the same hosting nodes t_i, t_{i+1} , as the sum of two α nodes exceeds one. Feasible configurations are pairs of consecutive nodes. As $n < m$, there will always be at least one edge ($\alpha \rightarrow \epsilon$) connecting a pair of unmatched nodes, as some segment of the hosting graph will not be used. Without loss of generality, let T_1 be the task whose α node is free, and task T_2 be the task whose ϵ node is free. The GCG cost function implies that the best strategy for T_2 is to move so that its ϵ node shares the hosting node with the α node of T_1 . Then, considering the set of strategies of all tasks, no matter what strategy T_i chooses, there will always be at least one free edge, and the task T_j whose ϵ node is free will be better off relocating. Since this holds for all possible strategies of all tasks, concluding that there is no equilibrium.

Similarly, in example (b) the interests of both players conflict. In this example each player asks for three different CPUs, with the additional requirement that the CPUs must be adjacent. In the configuration shown the top player would benefit by swapping the position of tasks 0.5 and 0.1, as its cost goes from 1.79 to 1.75. After this move, the best response for the other player is to swap its 0.5 and 0.3 nodes as his cost reduces from 2.25 to 2.20. At this point, this configuration is the symmetric of the original one, and the players will keep iterating forever.

Example (c) illustrates even another case, where the small task (node of weight 0.4) prefers to join the node of value 0.6 of the large task, which gives a cost of 0.4, the minimum possible for that player. However, the large player prefers to match its 0.3 node to the other player's 0.4. This gives a minimum cost of 1.43 for the large player. As both players can keep switching positions forever, the game never reaches an equilibrium.

Theorem 2. *Determining whether a GCG has a Nash Equilibrium is NP-Complete.*

Proof. Consider some instance of the 3-SAT problem, with m clauses and n variables as given below:

$$\begin{aligned} \phi &= C_1 \wedge \dots \wedge C_m \\ C_k &= l_{k1} \vee l_{k2} \vee l_{k3} \end{aligned} \quad , \quad \text{where} \quad \begin{aligned} l_{ki} &\in \{x_j, \neg x_j\} \\ x_j &\in \{x_1, \dots, x_n\} \end{aligned}$$

Then construct a GCG corresponding to the 3-SAT problem as follows (Figure 4-3 illustrates this construction).

Consider a set of $m + 1$ task graphs of the form $\epsilon \rightarrow \alpha$, such that $\alpha > 1/2$ and $\epsilon < (1 - \alpha)/m$. There is also a second set of tasks of the form $\delta \rightarrow 1$ with $\delta = 1 - m\epsilon$. For this collection of task graphs, it is only feasible to share allocations for ϵ nodes.

Consider a hosting graph with n pairs of nodes x_j and $\neg x_j$, with pairs of edges $x_j \rightarrow \neg x_j$ and $\neg x_j \rightarrow x_j$, for $1 \leq j \leq n$, corresponding to the variables of the 3-SAT problem. The graph also has m nodes C_k , for $1 \leq k \leq m$, corresponding to the clauses of the 3-SAT problem, as well as an edge per clause term l_{ki} connecting each of the variables x_j and $\neg x_j$ in a clause to the corresponding C_k node. By definition of 3-SAT there will be $3m$ of these edges. Finally, the hosting graph includes a ring of $m + n + 2$ nodes $t_1, t_2, \dots, t_{m+n+2}$. Set the costs and capacities of all nodes and edges to one.

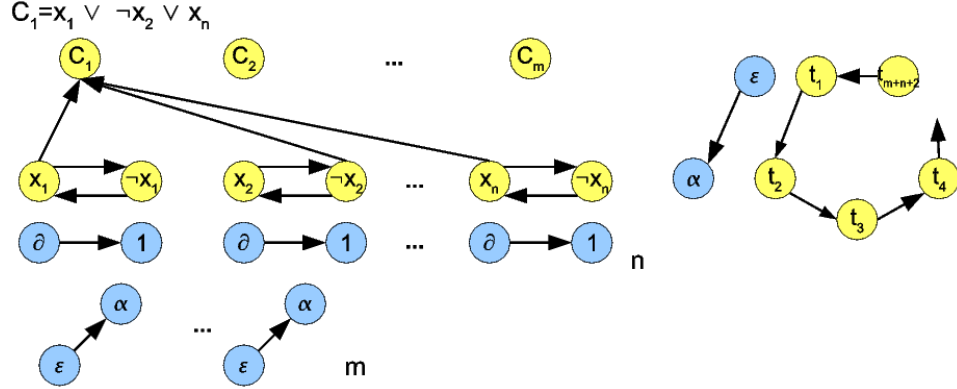


Figure 4.3: Construction used in proof of theorem 2.

Claim: If the 3-SAT problem instance is satisfiable, then there is a NE for the corresponding Colocation Game; otherwise there is no NE.

First, consider the case when the 3-SAT problem is satisfiable. Assign the GCG tasks as follows: There is at least one l_{ki} edge used in the true assignment per clause. Place an $(\epsilon \rightarrow \alpha)$ task from the variable node, to the corresponding clause node. Map the n $(\delta \rightarrow 1)$ tasks to the n pairs of $(x_j \leftrightarrow \neg x_j)$ nodes by matching the ϵ nodes to the δ nodes. Because 3-SAT is satisfiable, this assigns all the $\delta \rightarrow 1$ tasks and m of the $\epsilon \rightarrow \alpha$ tasks. Place the last $(\epsilon \rightarrow \alpha)$ edge in the ring. This configuration is a NE as there is no task with an alternative cost-reducing position.

Now consider the case when the 3-SAT problem is unsatisfiable. From the previous ring example, no configuration with more than one $(\epsilon \rightarrow \alpha)$ tasks in the ring is stable. So, place one in the ring and try to place the others outside. It is possible to place n on the edges $(x_j \rightarrow \neg x_j)$, but still would have m to place elsewhere. Because 3-SAT is not satisfiable, there are less than m edges $l_{ki} \rightarrow C_k$ and at least has to be placed in the ring, giving a configuration with no equilibrium. \square

Nash Equilibrium of PCG: Let $P = (\dots, p_i = P_i/U_i, \dots, p_m)$ be the vector of ratios of prices to utilizations for all resources. In addition, let $\mathcal{L}(P)$ be the sorting function $\mathcal{L}(P) = (p^{(1)}, p^{(2)}, \dots, p^{(m)})$, whose image is the vector or sorted components of P , *i.e.*, $p^{(1)} \leq p^{(2)} \leq \dots \leq p^{(m)}$. Consider the evolution of $\mathcal{L}(P)$ as a result of making a valid move. In particular, define P the vector before the move, and P' the vector after the move. The following lemma shows that the succession of vectors

$\mathcal{L}(P)$ is lexicographically ordered.

Lemma 3. *In a PCG, the ordered vector of unclaimed spaces after a valid move dominates the previous vector: $\mathcal{L}(P') < \mathcal{L}(P)$, where domination implies that there exists k such that $p'^{(i)} \leq p^{(i)}$ for all $i = 1 \dots k$, with strict inequality for k , $p'^{(k)} < p^{(k)}$.*

Proof. Assume task x_j is moving from a to b and that the sorted vectors of price ratios are

$$\mathcal{L}(P) = \left(\dots, \frac{P_a}{U_a}, \dots \right) \quad \text{and} \quad \mathcal{L}(P') = \left(\dots, \frac{P_b}{U'_b}, \dots \right)$$

By (4.5) it is the case that $P_b/U'_b < P_a/U_a$, which implies that P_b/U'_b will be either in the same position, or if it becomes smaller than one of the predecessors of P_a/U_a , it will move to the left. In either case, the element P_b/U'_b is strictly less than the corresponding element of $\mathcal{L}(P)$, and all other predecessors remain unchanged, therefore $\mathcal{L}(P')$ dominates $\mathcal{L}(P)$. \square

Theorem 3. *PCG converges to a Nash Equilibrium under better response dynamics.*

Proof. Lemma 3 defines a partial order in the set of vectors of price ratios. This partial order has one or more minimal elements⁷ and therefore the sequence of moves has to stop before, or at most when it reaches one of these minima. \square

Since PCG's NE may not yield the optimal packing, it might be useful to know the Price of Anarchy (PoA), *i.e.* the ratio of the worst-case cost at equilibrium to the social cost of an optimal solution. Figure 4.4 illustrates two examples, the first one for homogeneous resources and the second one for heterogeneous resources. Both examples are NE but the equilibria are not optimal. In the example of part a) all the resources have capacity one, the top configuration is a NE and the bottom configuration is optimal (OPT). The large processes have utilizations (weights) $1/2 < l \leq 2/3 - 2e/3$, so they cannot be colocated. The small processes have size $s = 2e/3$ and there are $1/e \geq 8$ of them. So the utilization when they are all in the same bin is $2/3$. Individually, none will move with the large objects, as this does not reduce their

⁷the minimal element of the lattice associated with this partial order is the vector corresponding to the minimum-cost bin-packing of all the tasks

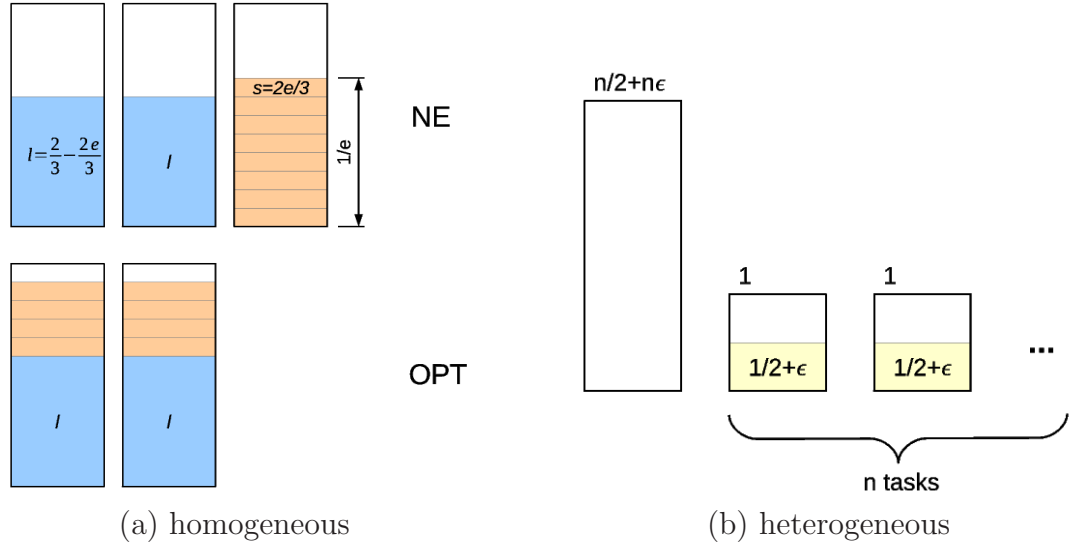


Figure 4.4: Examples illustrating PoA for PCG

cost, but when they all move with the large objects as in the bottom configuration, the cost of all players reduces and the social cost is minimum.

Theorem 4. *The price of anarchy for homogeneous PCG is $3/2$ while the price of the anarchy for heterogeneous PCG is 2.*

Proof. Assuming resources of equal price and capacity (all normalized to 1), a lower bound for an optimal colocation is the summation of all process utilizations, and an upper bound is placing one process per resource. If there are n processes this gives

$$\sum_{i=1}^n t_i \leq OPT \leq n \quad (4.9)$$

The PCG construction in figure 4.4(a)-bottom exemplifies a tight optimal colocation.

First, observe that if there were more resources with processes of utilization $1/2 < l \leq 2/3 - 2e/3$, then the cost ratio between NE and OPT would be $(|L|+1)/|L|$, where $|L|$ is the number of “large” processes. This ratio is maximized when $|L| = 2$. Alternatively, consider the case where there are more resources with low-utilization processes. This situation would not be an equilibrium as any process with utilization t_i in bin A would benefit by moving to bin B whenever $U_B + t_i \geq U_A$. Therefore, the equilibrium would occur when all low-utilization processes have filled as much as possible some of the resources. In the worst case, it would have been possible to

split such processes into two bins, with one large process of size $1/2 + \epsilon$ in each. This implies that a worst-case cost ratio of $3/2$ between NE and OPT.

As for the case where resources have non-uniform capacities, the PoA can be bounded, as illustrated by the example in Figure 4.4(b). This example shows a NE with cost n , where the optimal configuration has cost $n(1/2 + \epsilon)$, so this gives a lower bound $\Omega(2)$. Now, by contradiction assume an arbitrary collection of tasks such that $\sum t_i > n/2$ and whose PoA is greater than 2. By the definition of PoA and using (4.9) this would imply $n / \sum t_i > 2$, which contradicts the fact that the initial set of tasks has $\sum t_i > n/2$. Therefore 2 is a tight bound on the PoA. \square

It is also possible to bound the number of moves it takes for the *Colocation Game* to reach a NE in the homogeneous case, as stated in the following theorem:

Theorem 5. *When having a minimum threshold for executing a move, the PCG converges to a NE in $O(n^2)$ iterations.*

Proof. Assume without loss of generality unitary capacities and prices. Let V be the vector of unclaimed spaces

$$V = \mathbf{1}_n - U = (v_1, \dots, v_n)$$

then consider the following potential function

$$\phi(V) = v_{(1)} + \dots + nv_{(n)} \tag{4.10}$$

where n is the number of resources, and $v_{(i)}$ are the components of $\mathcal{L}(V)$. Define also the minimum allowed improvement ϵ as the threshold in the decrease of unclaimed space necessary to make a move. If the change is below this threshold, the move will not be performed, as the gain for the user is negligible. Then, when an object moves from j to i , the potential of resource i decreases by at least $\epsilon(n - i + 1)$ and the potential of j increases at least by $\epsilon(n - j + 1)$. The minimum change in potential is

$$\begin{aligned} \Delta\phi(V) &= \epsilon(n - i + 1) - \epsilon(n - j + 1) \\ &= \epsilon(i - j) \end{aligned}$$

which is minimal when moving between adjacent resources, in which case is ϵ .

At the beginning the potential is at most

$$\phi(1, \dots, 1) = \sum_{i=1}^n i = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$$

An upper bound on the number of moves m would then be the number of moves to go from the maximum potential to just below the threshold

$$\begin{aligned} m\varepsilon &\geq \frac{n^2}{2} - \varepsilon \\ m &\geq \frac{1}{\varepsilon} \left(\frac{n^2}{2} \right) - 1 \\ m &= O(n^2) \end{aligned} \tag{4.11}$$

□

Now consider multidimensional PCG: According to (4.8), valid moves are defined by the ratios of price to utilization of the resources (a scalar quantity). Consequently, the exact same construction used for the unidimensional case describes the dynamics of price ratios vector for multidimensional PCG, yielding the following theorem.

Theorem 6. *Multidimensional PCG converges to a Nash equilibrium under better response dynamics.*

Lower-bounding the worst-case cost paid by a player: As defined before, player i 's cost when placed in resource R is

$$P \cdot \frac{v_i}{\sum_{j \in R} v_j}$$

The worst-case for a player is when there is only one user (Figure 4-5a) and in this case the cost is P . The best case is when the player only has to pay for its portion of the volume, and the volume taken by other players is maximum (Figure 4-5b). In this case the player has to pay

$$P_{best} = P \cdot \frac{v_i}{v_i + \prod_{k=1}^d (c_k - x_k)}$$

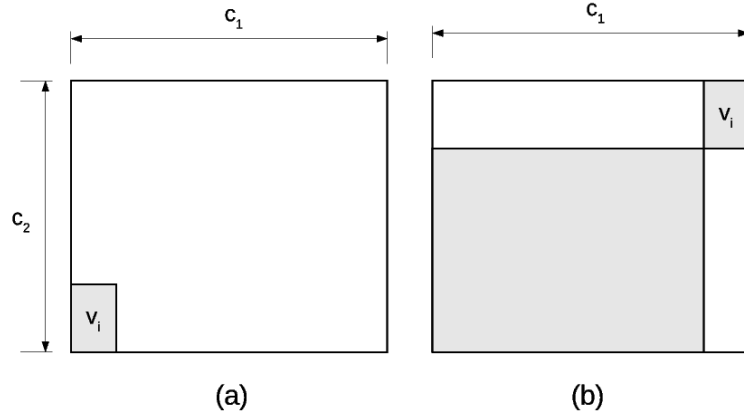


Figure 4.5: Worst and best cases for a player's cost

hence the ratio of the worst-case cost to the best case is:

$$\frac{worst}{best} = \frac{v_i + \prod_{k=1}^d (c_k - x_k)}{v_i} = 1 + \frac{\prod_{k=1}^d (c_k - x_k)}{\prod_{k=1}^d x_k}$$

In the continuous case it is unbounded, but considering a discrete representation of b -bits along each dimension, with a minimum permissible allocation of 1 unit, and a resource capacity along each dimension of $2^b - 1$, then the worst case ratio for d dimensions can be bounded by

$$\frac{worst}{best} = 1 + \frac{\prod_{k=1}^d (2^b - 2)}{\prod_{k=1}^d 1} = 1 + O(2^{bd}) \quad (4.12)$$

A practical consideration resulting from this analysis is that given the heterogeneity of the resources and of the task sizes, using a large number of bits to represent capacities and demands would severely hurt the worst-case cost for a player. Instead, subdividing the resources into classes and assigning tasks to the smallest class that can accommodate it, makes it possible to use a reduced number of bits per class. This subdivision also makes sense from the point of view of the cloud-resource provider, as it is common to acquire machines in homogeneous batches, with each batch belonging to different hardware generations whose capacity typically increases

with improved technology. Having various classes of homogeneous resources also brings the additional benefit that the price-of-anarchy is smaller compared to the case with heterogeneous resources.

To conclude this section, some equilibrium results for PPCG are presented.

Theorem 7. *On the one hand, PPCG with uniform processes converges to a NE under better response dynamics. On the other hand, better response dynamics for PPCG with non-uniform processes may cycle.*⁸

Proof. In uniform PPCG, consider a better response of task T_i . Since all of T_i 's node utilizations are the same, and since one cannot colocate any pair of these nodes on the same hosting node, the better response of T_i can be realized by a sequence of moves involving only one node. Therefore, by applying the same potential function as in Lemma 3 and Theorem 3 it is possible to conclude that the better response dynamics converges. Examples (b) and (c) in Figure 4-2 illustrate the case in which non-uniform PPCG cycles. \square

4.3 PCG: Implementation

4.3.1 System Architecture

The implementation of PCG requires the development of two sets of services. The first would support *strategic* (decision-making) functionalities for the players, whereas the second would support *operational* functionalities of the system. These services can be implemented as two autonomous agents: a *user agent* implements the per-user functionality and a *server agent* implements the server-side functionality. Both classes of agents rely on a global distributed directory. *Server agents* register themselves in the directory and *user agents* query the directory to find one or more servers when needed. During the strategic phase, the user agent's interaction is the following: Pick a server from the server directory, contact the corresponding server agent and submit the user's bid, which contains the request vector X_i . The server agent

⁸This theorem can be extended to the multidimensional version of PPCG.

accepts the bid if it is a valid move (as defined in 4.8) and informs the user agent of the result. The user agent is free to repeat this process as many times as desired, and by the convergence property of PCG (Theorem 3) this process will reach a NE in at most $O(n^2)$ interactions (Theorem 5). Replacement moves are implemented by the server agent by informing the user agent that its bid has been superceded. At this point it is up to the user agent to bid again on another server. An accepted bid implies a commitment by the user agent to make a payment for its fair share of the server according to the cost function (4.7). Clearly, the bidding process must implement a secure protocol to authenticate users and enforce the commitments to pay. This also ensures that the user agent only has one active bid at any point in time as multiple bids would result in multiple payments.

Operationally, at the start of an epoch, it may be necessary to reconfigure the system to be consistent with a new colocation configuration. This requires the migration of a subset of the user processes from one set of cloud resources to another. Notice that this migration does not involve strategic choices by the players (*i.e.*, it is not a game), but rather a decentralized optimization [HHK⁺01], whereby it is beneficial to all constituents if the transition is done in the most efficient manner, and also if its execution does not impose significant (or even measurable) degradation in the service delivered to cloud users. This is an example of an operational service, reminiscent of the live migration problems for virtual machines [CFH⁺05] (*e.g.*, VMware’s VMotion and Xen live migration).⁹ Other operational services would include typical metering, accounting, and policing services.

⁹Notice that separating the strategic and operational aspects of the implementation allows us to avoid the redundant (and potentially very expensive) migrations resulting from a “literal” implementation of a move in PCG.

4.3.2 Better Response Heuristics

A practical distributed implementation of the *Colocation Game* should rely only on local information available to the user agent and a polynomial algorithm for computing its better response. When a *user agent* is about to make a move, it queries the distributed directory to get a random server pointer. It submits its request vector X_i to the corresponding *server agent*. If adding the process to the resource results in a cost reduction, then the server replies accepting the bid. This produces a *placement move*. On the other hand, if adding the process is not feasible, it may be still the case that a subset of the other processes along with the one making the bid results in a lower cost to all the members of the set, resulting in a *replacement move*. Determining if such subset exists corresponds to solving a knapsack problem [CLRS01]: Find the subset of processes that maximizes the utilization of this resource. If a cost-reducing solution is found, the replacement move is executed, otherwise the process has to wait until its next turn to try again.¹⁰

The knapsack problem itself is NP-complete, but there is a dynamic programming algorithm for solving it in pseudo-polynomial time for reasonable size instances. Given that in practice the number of processes assigned to a resource is unlikely to exceed a few hundreds, the Dynamic Programming solution of the Knapsack Problem (DPKP) would be tractable. In addition to DPKP, other branch & bound techniques using either breadth-first search (BFS) or depth-first search (DFS) are considered. Branch & Bound BFS and DFS limit the degree and the depth of the search to limit the amount of time spent finding a better response. The remainder of this section discusses the details about DPKP, BFS, and DFS as well as the local better-response strategy.

¹⁰To reduce the total number of moves, processes may adopt a cost threshold heuristic, whereby a process will only adopt a move if the resulting cost improvement is at least $p\%$ of its current cost (or is at least some fixed absolute threshold).

Dynamic Programming Solution of the Knapsack Problem (DPKP)

With reference to equation (4.8), notice that the problem of minimizing the cost incurred by a task for packing its process in a resource reduces to the problem of maximizing U'_b , or equivalently minimizing the slack (unused capacity) of a resource. For this analysis, define a workload as a set of tasks (processes) $W = \{T_k, \dots\}$, and the slack as $s(W) = R - \sum_{T_i \in W} T_i$, where R is the capacity vector of the resource. The quantity to minimize is the volume of the slack $v(W) = \prod s(W)$. For the construction of the dynamic programming solution, assume that demands and capacities are discrete b-bits quantities, in a d-dimensional space. This implies that there are 2^b values per dimension, yielding a total of 2^{bd} discrete volumes.

The key observation in defining DPKP is that the best solution that includes task T_i is the union of $\{T_i\}$ and the best solution for the resource of capacity $R - T_i$. Considering one additional task at a time, it may be the case that either the best solution for some capacity R is the same as when considering only tasks T_1, \dots, T_{i-1} , or the alternative solution for $R - T_i$ union $\{T_i\}$. Therefore the overhead of the optimal solution can be computed using the recurrence:

$$A(i, j) = \min \{A(i-1, j), v(W_{(i-1,k)} \cup T_i)\} \quad (4.13)$$

where the column indexes (j, k) are over an enumeration of the capacity vectors whose increments are the discrete capacity steps, so in (4.13), k is the index of the column corresponding to the capacity of column j minus task T_i :

$$k = \text{column}(R_j - T_i)$$

As a boundary condition, for cases where $R_j - T_i \leq 0$ on any of its dimensions, the cost is ∞ , same for $A(0, j)$. Observe that the rows need not be ordered. All that is needed when computing row i is to have all the values of row $i - 1$. Therefore

```

function searchBetterMove(Set of tasks T)
    A(0,:) = Infinity
    tasksPerColumn(0,:) = {empty}
    maxColumn = 2^(b*d) // Bits times Dimensions
    foreach i in T
        for j=0 to maxColumn
            residualVolume = volume(j)-volume(i)
            if residualVolume>0
                k = columnOf(residualVolume)
                cost = volumen( tasksPerColumn(0,k) Union i )
                if cost<A(0,j)
                    tasksPerColumn(1,k) = tasksPerColumn(0,k) Union i
                    A(1,j) = cost
                endif
            endif
        endfor
        A(0,:) = A(1,:)
        tasksPerColumn(0,:) = tasksPerColumn(1,:)
    endfor
endfunction

```

Figure 4-6: Pseudocode for the DPKP Better Response Algorithm

the space requirement is limited to keeping in memory the current and the previous rows. If there are d dimensions and b bits per dimension, the total space required is $O(2^{bd})$. As for time complexity, the algorithm finishes when it reaches the last column of the last row, therefore if m is the number of tasks, the time is $O(m2^{bd})$. This solution suffers from the *curse of dimensionality* for large values of d or for high granularities. Figure 4-6 gives the pseudocode of DKPD.

Branch and Bound Better Response Heuristics

Figure 4-7 shows the pseudo-code for the branch-and-bound better-response heuristic. It takes two parameters *degree* and *maxDepth* that limit the portion of the search space explored. If the process fits in the given resource, the algorithm returns immediately. Otherwise, the algorithm searches for the minimal set of tasks that need to be replaced in order for the new task to fit. In this pseudocode, *list* is a data structure that may be implemented as a FIFO queue to give a BFS search, or as a LIFO queue to give a DFS search. The algorithm keeps track of the best solution

found so far, and when it is done (either because it was able to explore the full search space or because it reached the bounds established by *degree* and *maxDepth*) it returns its current best solution.

Local Better-Response Procedure

This section summarizes the complete procedure for a process to find a better-response move using any of the previous techniques to solve to the subjacent knapsack problem. Let k denote the process whose turn it is to make a better-response move. Let P_0 and U_0 be the price and utilization of the resource to which player k is currently assigned.

1. Player k randomly picks a resource j of capacity R_j and price P_j . Let V_j denote the set of tasks currently assigned to resource j .
2. Let $S \subseteq V_j$ be the solution of the knapsack problem (obtained using any one of the approaches presented above), where the knapsack capacity is $R_j - X_k$ and the set of tasks is V_j . Let v_i be the volume of task i . A valid move exists if

$$\begin{aligned} \sum_{i \in SU\{k\}} X_i &\leq R_j \\ \sum_{i \in SU\{k\}} v_i &> \sum_{i \in V_j} v_i \\ \frac{P_j}{\sum_{i \in SU\{k\}} v_i} &< \frac{P_0}{U_0} \end{aligned}$$

The first condition for a valid move is the feasibility condition, which is implicit in any knapsack solution. The second condition underscores the total utilization increase, assuring that the cost of the tasks that will remain assigned to resource j will decrease. The third condition is the cost reduction for player k .

```

function initialize(Task T, Resource R)
    need = R.utilization()-(R.capacity()-T.volume())
    list = {}
    tasksToReplace = {}
    if need<0
        // Move found no task needs to be replaced
        return tasksToReplace
    for i=1 to degree
        chosen = random not chosen task from R.tasks()
        node.parent = null
        node.task = chosen
        node.sumVolume = chosen.volume
        node.depth = 1
        list.add(node)
    endfor
    bestSolution = search(list, T, R)
    if (bestSolution != null)
        tasksToReplace = set of tasks in the chain from
            bestSolution to the root
        return tasksToReplace
    else
        return "no better move found"
    endif
endfunction

function search(list, T, R)
    bestCost = infinity
    bestSolution = null
    while( ! list.isEmpty() )
        node = list.remove()
        newVolume = R.sumVolume() - node.sumVolume + T.volume()
        newCost = R.price()*T.volume()/newVolume
        if (newVolume>R.sumVolume() and newCost<T.cost()
            and newCost<bestCost)
            bestCost = newCost
            bestSolution = node
        else
            if (node.depth<maxDepth)
                for i=1 to degree
                    chosen = random not chosen task
                        from R.tasks()
                    child.parent = node
                    child.task = chosen
                    child.sumVolume = node.sumVolume
                        + chosen.volume
                    child.depth = node.depth+1
                    list.add(child)
                endfor
            endif
        endif
    endwhile
    return bestSolution
endfunction

```

Figure 4-7: Pseudocode for the Branch-and-Bound Better-Response Heuristics

3. If player k is able to identify a valid move, then it replaces the set of tasks $V_j \setminus S$ on resource j .¹¹

4.4 PCG: Experimental Evaluation

To explore the potential from using colocation games as the underlying mechanism for enabling rational, selfish users to make cost-effective use of a distributed infrastructure available through an independent provider who has no economic incentive to do so, this section presents a summary of a series of experiments for variants of the process colocation game, which are natural candidates for resource management in distributed/cloud computing environments.

Data Sets and PCG Variants: These experiments were conducted using both, synthetically-generated, and trace-driven (PlanetLab) workloads. For each one of them, the unidimensional and multidimensional variants of PCG were simulated under both a homogeneous and heterogeneous resource pool.

The *synthetic workloads* give the flexibility of varying the number of resources, number of tasks, and dimensionality. More importantly, they make it possible to experiment with workloads for which the socially-optimal solution is known. This is done by repeatedly fragmenting the full capacities of a set of resources and then using the resulting fragments as a representation of the utilization (demand) of a set of tasks. By construction, it is known that a “perfect” colocation exists (with every resource being fully utilized). As the initial configuration for the experiment, the resulting tasks are scrambled in a much larger set of resources.

The *trace-driven workloads* were constructed using publicly available PlanetLab traces of CoMon [PP06, web]. These traces contain snapshots of PlanetLab server

¹¹As a result of a replacement move, the set of replaced tasks ($V_j \setminus S$) are assigned to a new (empty) resource. This will necessarily increase their costs, possibly triggering these tasks to execute their own better-response moves later on.

capacities as well as the utilization of the slices assigned to the various tasks colocated on each server. The main advantage of this dataset is that it gives the distribution of a set of real task utilizations on a fairly large scale. Its disadvantage is that one cannot compare the configurations resulting from the collocation game to socially-optimal configurations – the latter being infeasible to compute due to PlanetLab’s scale. Thus for these workloads, the worst/best cost ratios resulting for a series of runs of the same game instance are reported. In PlanetLab, server capacities as well as slice utilizations (corresponding to the utilizations of a slice’s CPU, memory, uplink, and downlink) are multidimensional. This makes PlanetLab an example of an infrastructure on top of which a multidimensional PCG game may be implemented. As with synthetic workloads, the PlanetLab experiments considered both homogeneous and heterogeneous resource models.

Player Response: As computation of a player’s best response is an NP-complete problem, the experiments compare the various solution techniques from Section 4.3, namely: (1) A dynamic-programming knapsack solution (DPKP), (2) A depth-first search (DFS) with branch-and-bound pruning, and (3) a breadth-first search (BFS) also with branch-and-bound pruning.

Metrics: The metrics used to characterize the PCG game dynamics were: (1) the *colocation ratio* which is defined to be the ratio of average/optimal social cost (for synthetic workloads) and of worst/best social cost (for trace-driven workloads); (2) the total number of move *trials* until an equilibrium is reached; and (3) the total number of actual *moves* until an equilibrium is reached. The colocation ratio characterizes the (in)efficiency of the colocation resulting from playing the game, and is bounded by the PoA, as derived analytically for unidimensional PCG. The number of trials gives an insight on the total time it takes for the game to reach an equilibrium.

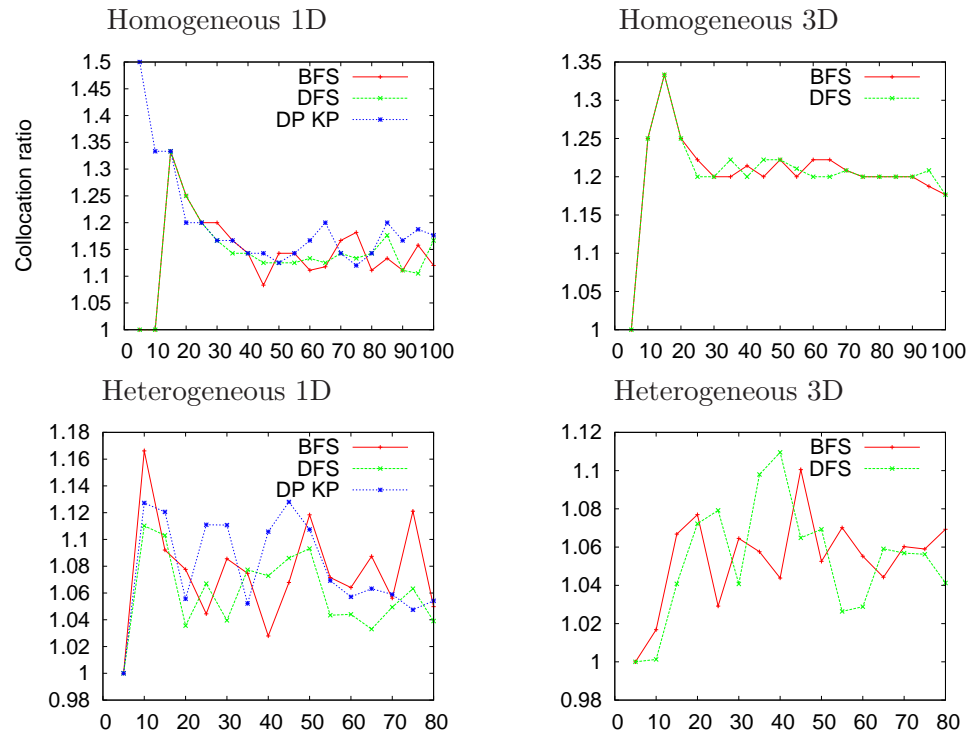
To compute all three metrics, the game runs until the system reaches an equilibrium. Because of the heuristic solution to the better response problem, this equilibrium is not necessarily a NE, but a relaxed notion of equilibrium where no player was able to find a better response with bounded computational resources. The metrics reported in the experiments below correspond to this equilibrium.

Colocation Efficiency for Synthetic Workloads: Figure 4-8 shows the median (over 100 samples) of the colocation ratios for synthetically-generated workloads in both homogeneous and heterogeneous settings. Recall that in a homogeneous (heterogeneous) setting all resources are (not) of equal capacities. These results show that the PoA (of $3/2$ for homogeneous cases and 2 for heterogeneous cases) for the colocation for most cases. A few exceptions registered in the worst-case plots are attributed to the approximate better-response computation and to the threshold for detecting an equilibrium.

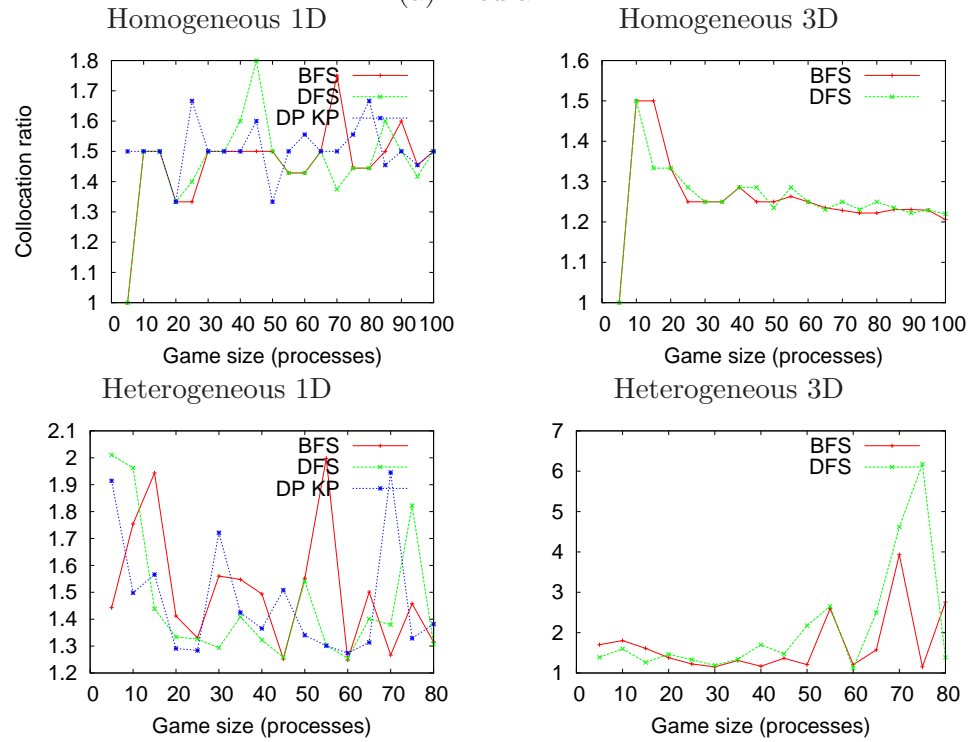
The results in Figure 4-8 show that the colocation ratio tends to decrease (*i.e.*, better efficiency) as the number of players increases, which bodes well for large-scale deployments. Also, these results show that colocation efficiency was basically independent of the better-response heuristic in use. It is noteworthy to mention that the branch-and-bound BFS and DFS heuristics were much faster than the dynamic programming (DPKP) heuristic. Indeed, DPKP was not able to handle many of the 3D and PlanetLab instances of PCG (hence the omitted DPKP results in Figure 4-8 for 3D cases).

Comparing the results for homogeneous versus heterogeneous settings, note that the median colocation ratio in the latter setting was only slightly larger.

Colocation Ratio for PlanetLab Workloads: Figure 4-9 shows the median colocation ratio (ratio of worst/best social cost) using the task specifications derived from the PlanetLab traces. These results imply a relatively small colocation ratio,



(a) Median



(b) Worst-case

Figure 4-8: Collocation ratio for synthetic workloads.

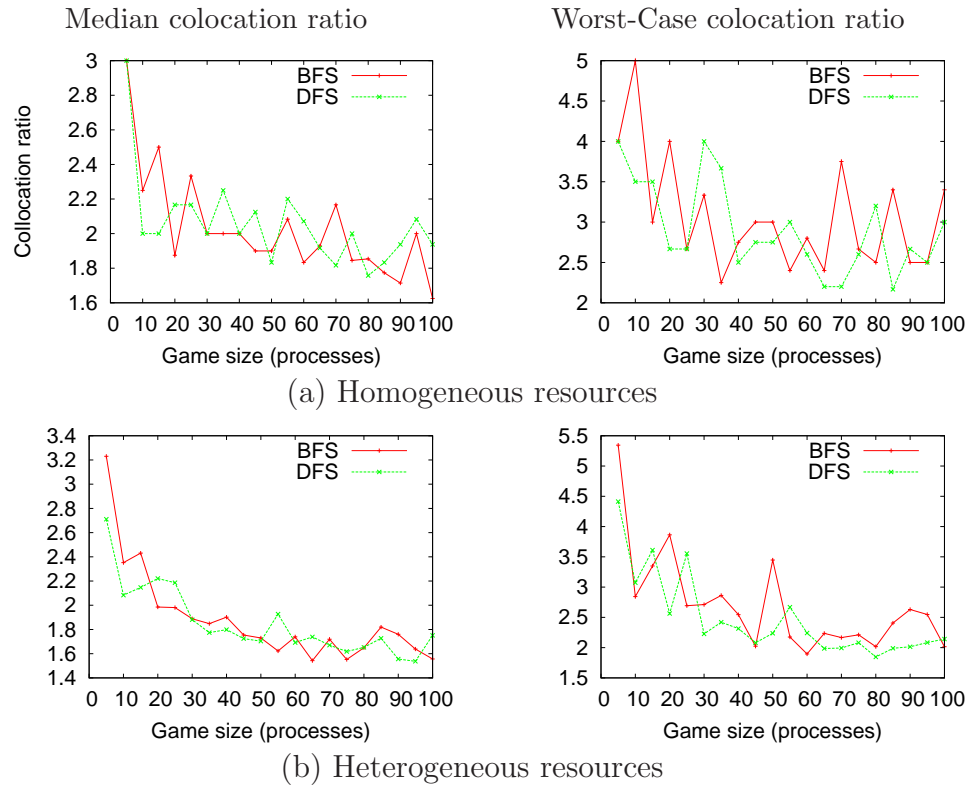


Figure 4-9: Collocation ratio for PlanetLab workloads.

which as with synthetic workloads seems to be independent of the better response heuristic used. As shown in Figure 4-9, the worst-case collocation ratios were not too far off.¹²

Convergence Speed and Overhead: Figures 4-10 and 4-11 show the number of moves it takes to reach equilibrium. They indicate that the number of moves is directly related to the number of tasks in the system and fairly independent by the heuristic used to compute the better response. The relation is essentially linear for unidimensional PCG and follows a power law for higher dimensionality PCGs.

Figures 4-12 and 4-13 show the number of trials to reach equilibrium. Similar to the number of moves, the number of trials follows a power law dependent on the dimensionality of the problem, almost linear for 1D settings.

¹²Note that the abundance of small tasks in the trace significantly improve the chances of achieving better placements.

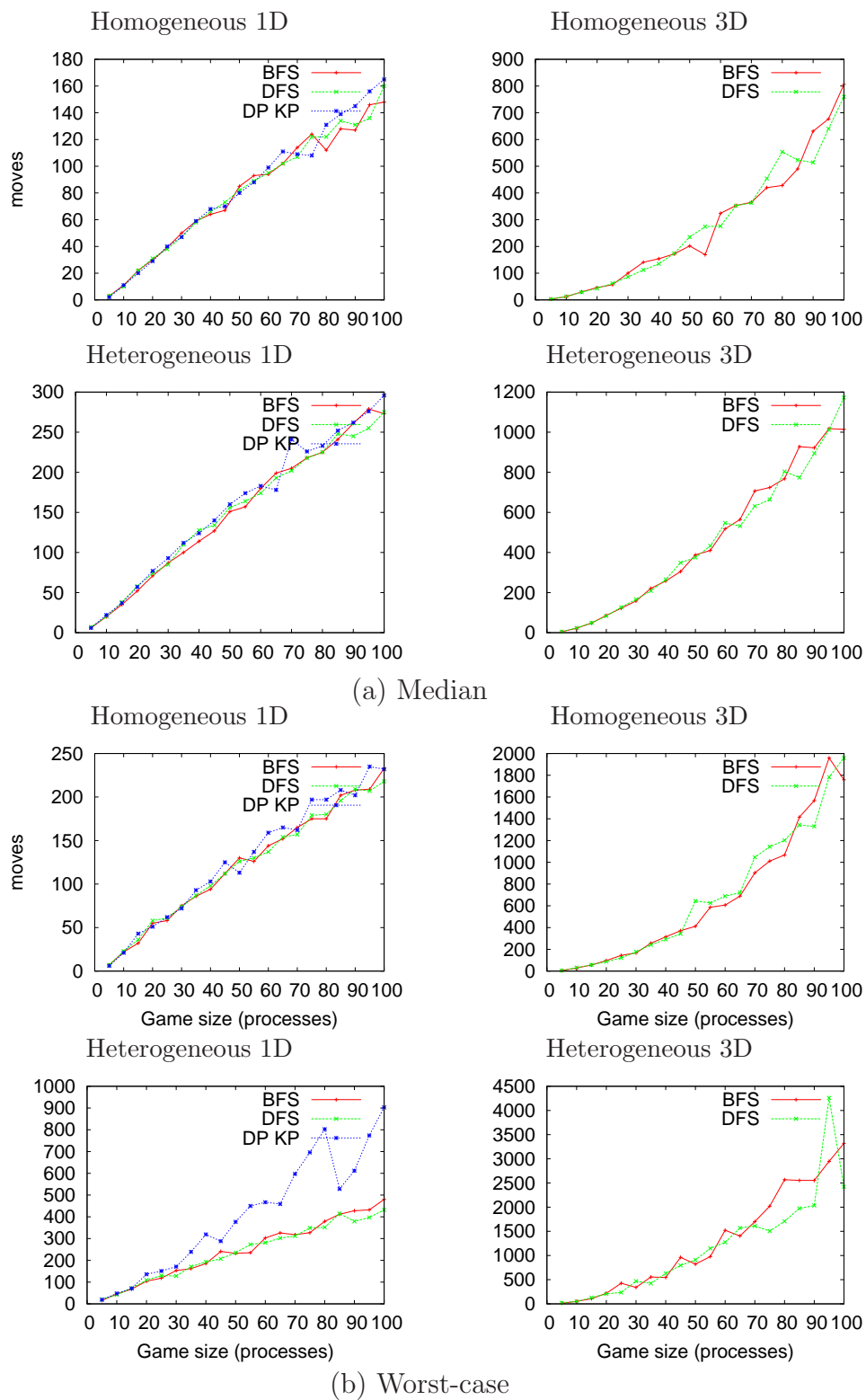


Figure 4-10: Number of moves until NE for synthetic workloads.

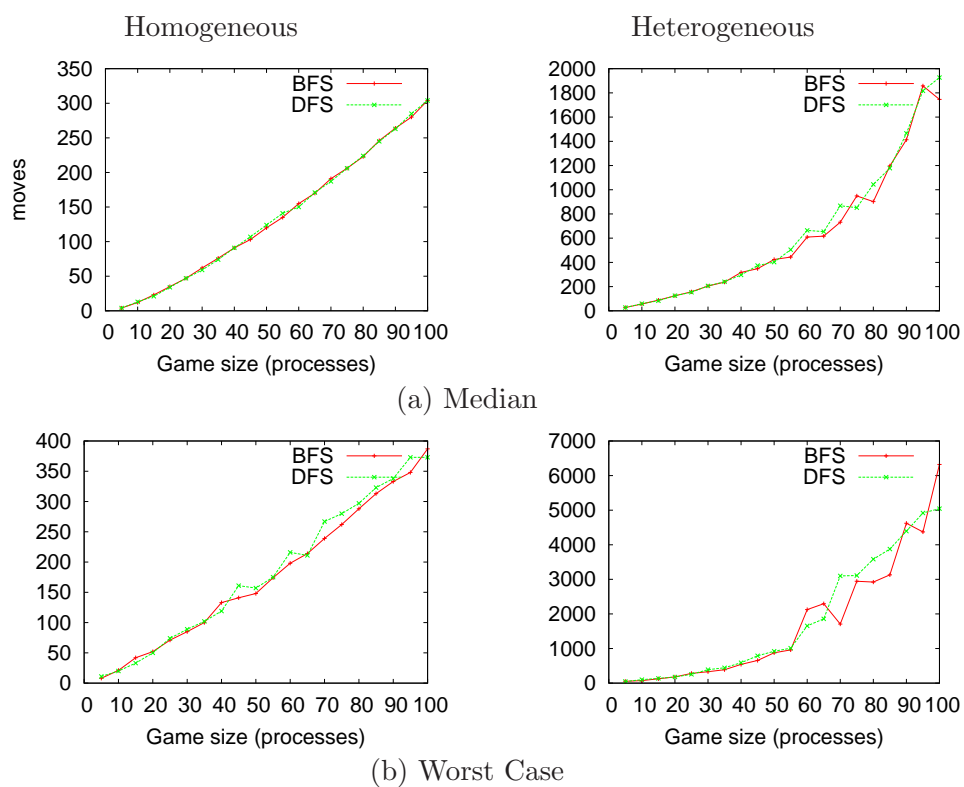
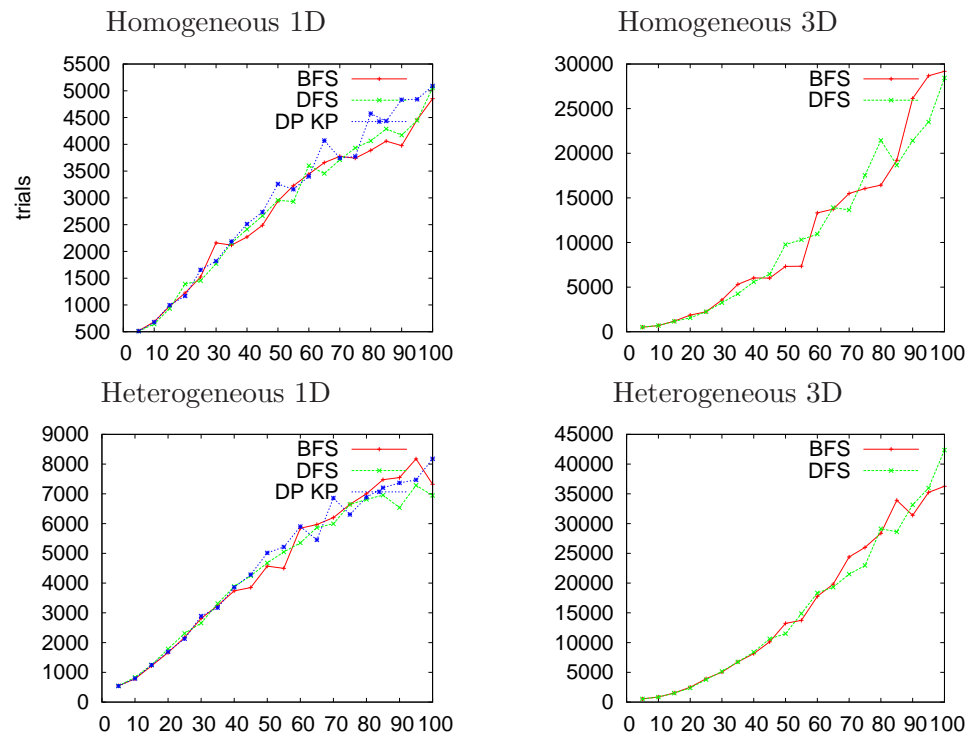
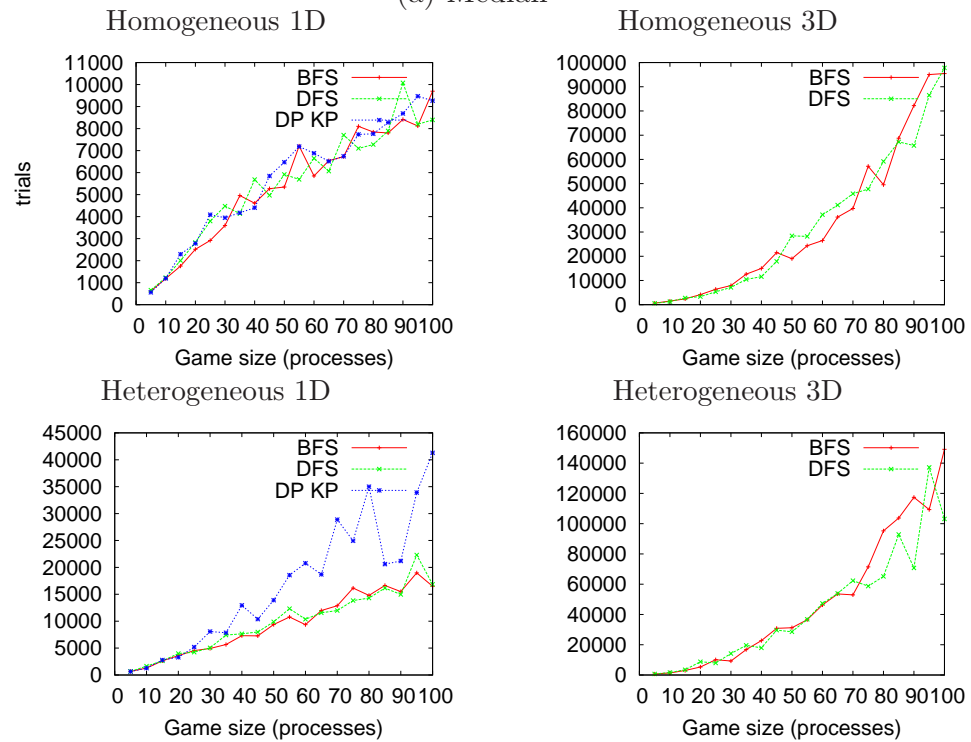


Figure 4-11: Number of moves for PlanetLab workloads.



(a) Median



(b) Worst-case

Figure 4.12: Number of trials for synthetic workloads.

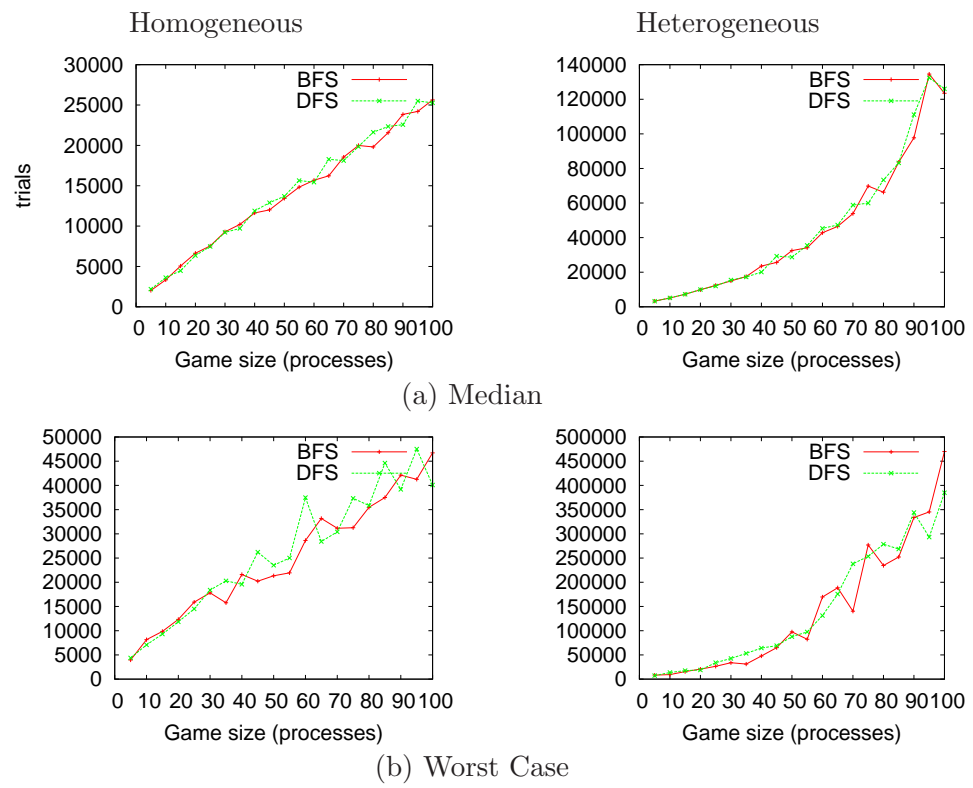


Figure 4-13: Number of trials for PlanetLab workloads.

Another question explored experimentally was the problem of assigning turns to the players. Various heuristics were explored: 1) random, chooses the player who is going to make the trials uniformly at random, 2) round-robin, all players go around taking turns, and 3) worst-located first, players who are currently paying higher overhead costs get more chances to move. To evaluate the effect of the selection heuristic on the convergence dynamics, figures 4.14 and 4.15 show the number of moves and trials under the various strategies. In general the effect is minimal suggesting that the convergence speed is independent of the selection policy.

4.5 Related Work

4.5.1 Resource Allocation Using Micro-economic Mechanisms

When multiple users compete for a set of resources, the question of which applications users get allocated and which applications users get denied (or postponed) arises. Auctions are a micro-economic mechanism that handles this question on the assumption that there is an authority running the auction and enforcing its outcome. One concrete example where auctions are being used is Bellagio [ACSV04], which specifically uses a combinatorial auction mechanism with the goal of maximizing the social utility (the sum of the utilities for the users who get resources allocated). Users express their valuations for different sets of resources (*e.g.* the various mappings found by SWORD) in form of *bids*. One particularly interesting auction mechanism is the VCG auction [AM04] because it maximizes the social value and makes truthful bidding a dominant strategy. The VCG mechanism has its limitations though [San96]: it is not budget balanced, it is susceptible to collusion and the implicit WDP is NP-Hard. Instead, Bellagio relies on a thresholding auction mechanism called SHARE [CNA⁺04] for clearing the auction. In order to enforce the outcome of the allocation mechanism (the auction or any other) in a distributed

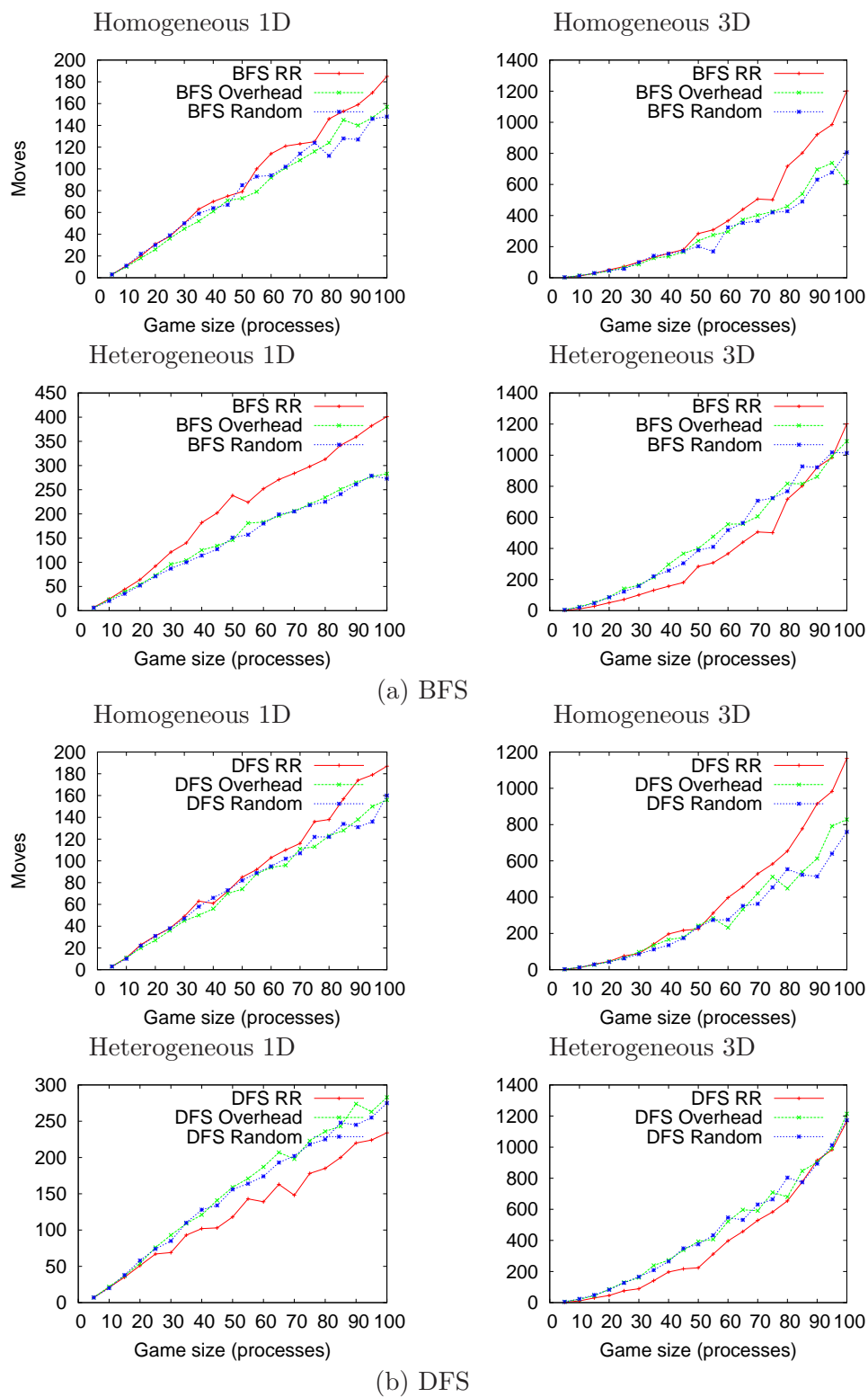


Figure 4-14: BFS & DFS Median number of Moves – Synthetic dataset

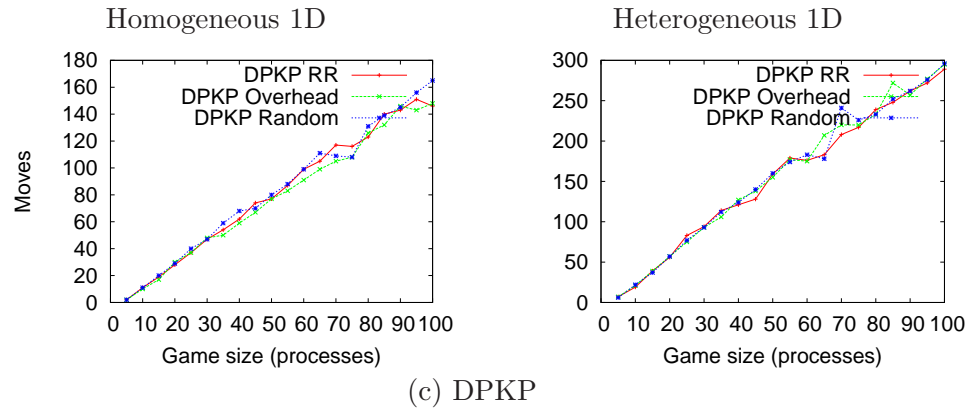


Figure 4-15: DPKP Median number of Moves – Synthetic dataset

and federated environment, where the management of the resources available at each site resides with the site’s administrators, a secure delegation mechanism is required. SHARP [FCC⁺03] is an example of a mechanism that provides this functionality.

Practical experience with auction-based mechanisms has revealed other vulnerabilities in the form of user behaviors that manipulate the outcome of the auction. In particular AuYoung *et al* [ABC⁺09] point out the following ones:

1. *Underbidding*: when users know that the overall demand is low they can drive the prices down.
2. *Iterative bidding*: users keep changing their bids, which may lead to a never ending bidding process, or make the auction take too long, which is problematic when resources are perishable (remaining idle).
3. *Rolling window manipulation*: when users may bid for resources at various points in time. It may result in starvation of some users and suboptimality in the utility of the system over time.
4. *Auction sandwich attack*: may occur when users are allowed to bid for resources at different points in time. Given limited information, the system computes a

locally optimal solution, which may differ from the global optimal were all the requests known at the beginning.

Another example of a system to handle the allocation for multiple users is Tycoon [LRA⁺05]. In this case the assumption is that resources are shared under a best-effort model and each user should be allocated resources in a proportionally fair manner. Resources are federated in the sense that allocation decisions are local to each one of the resources, thus avoiding the combinatorial nature of the WDP, at the expense of giving up optimality. In this system users place bids on the different resources they need. The fraction of resource allocated to one user is the proportional share according to the total bids in the system. For this reason it is a best-effort system: there is no guarantee applications will receive some desired fraction of the resources. The bidding process is continuous in the sense that any user may modify/withdraw its bid at any point in time, and the allocation for all the users is adjusted according to the new bid-to-total ratios. Users have finite budgets, therefore their best strategy is to maximize their utility (a linear combination of the utilities derived from each resource) subject to the available budget.

4.5.2 Resource Allocation Using Game-Theoretic Mechanisms

Suri *et al* [STZ04] present an analysis of P2P system where users follow a selfish strategy to establish connections to peers. This analysis models the system using a congestion-game where the time to complete a transfer increases as the load on a peer increases. Their analysis shows the existence of NE and gives bounds on the PoA for both, the case of all peers having the same speed, and the case of peers with heterogeneous speeds.

The formation of overlays between selfish peers is a problem of interest in P2P applications. For example Laoutaris *et al* [LSBB07] make a characterization of the

resulting overlays when the peers selfishly minimize the cost of reaching all other nodes in the network.

The work by Chen and Roughgarden [CR06] is the closest in spirit to the *Colocation Games*. In this work, the goal is to allocate source-destination flows on a shared network. Each flow has a weight and the cost of the link is split in proportion to the ratio between the weight of a flow and the total weights of the flows sharing the link. This work shows that NE does not always exist, but approximated NE¹³ does exist.

4.6 Summary

Service providers will do what is within their capabilities to optimize the operation of their services, generally meaning to maximize their utility.¹⁴ However, when the service providers are bound by SLAs or the granularity of their reservations is coarse, this creates the opportunity for users to look for alternatives that optimize their own benefit, which is the situation modeled by the *Colocation Game*.

Assuming a generic cost-sharing model (the Shapley cost), it was shown that GCG, the most general case of the *Colocation Game*, best-response dynamics do not always converge to a NE, and determining if a NE exists is an NP-Complete problem. In practical terms this is a negative result as for any given workload (collection of user applications) not reaching an equilibrium means that the system will spend its time readjusting to the user's requests to change their mappings in response to the changes made by other users, all of which is overhead for the system.

This chapter also considered several special cases for which it was shown that NE always exists: the Process Colocation Game (PCG), the Multidimensional Pro-

¹³A configuration is an α -approximate NE if no player can reduce its cost by more than an α multiplicative factor.

¹⁴Taking the utility in a very general sense, and not necessarily meaning the revenue for the provider. For example non-for-profit infrastructures such as Emulab, PlanetLab, and Mirage look forward optimizing the value of the testbed for their users, rather than maximizing their profit.

cess Colocation Game (MPCG), and the homogeneous Parallel Process Colocation Game (PPCG). Even though they are much simplified versions, there are practical applications for which they constitute a feasible resource management framework, where allocation decisions are delegated to the end-users, can be easily implemented in distributed and federated environment, and extend to online settings, where applications may arrive and leave over time and the system readjusts itself to until it reaches a new NE. It was also shown that the PoA of using the game as an allocation system is bounded and in fact competitive with the approximation ratio for the bin-packing problem, meaning that the allocation decided by the game is comparable to what could be obtained by a centralized approximated solution. Extensive experimental evaluation showed that in fact this upper bound is rarely reached, meaning that in practice the colocation ratio is much less than the PoA. This is the case especially when the number of users is not very small – exactly the scenario expected on large infrastructures where many users (in the order of hundreds for typical testbeds, and much larger for Cloud service providers) share the infrastructure.

Chapter 5

The Trade & Cap Marketplace: Incentives for Load-Balancing

The network embedding problems studied in chapters 3 and 4 emphasized the *spatial* resource mapping considerations, meaning that all the resources were to be assigned at the same time, to be used by the applications over the same time interval. On the other end of the spectrum are problems, where the goal is to control the *temporal* allocation of resources. Doing so is important in many practical cases, exemplified below:

1. For Infrastructure as a Service (IaaS) providers, the efficiency of datacenters is heavily dependent on the utilization level [BH09, GHMP09]. For example, Amazon's recently introduced EC2 Spot Instances [Ama09] service tackles this problem by introducing a supply-and-demand driven pricing so that "elastic instances" (those who are flexible about when they can be run) get an incentive to fill in underutilized periods and reduce the pressure on periods of high demand.
2. When a set of network users share a common link, as is the case for broadband users connected to the same ISP, or the users within an enterprise/campus sharing a dedicated access link, the performance of each application is heavily dependent on the utilization level of the common link. For example, queueing delay increases rapidly with the utilization level and adds to the response

time observed by end users. The normal practice is for users to execute their network tasks immediately, which translates into high variability of network utilization and poor performance during periods of high utilization. It has been pointed out [LR08, LSRS09] that there are applications that need not to be executed right away, but can be perfectly executed at a later time, as for example P2P file transfers, network backups, and bulk data transfers. For this reason they are called Delay-Tolerant (DT) and significant savings can be obtained by scheduling their execution appropriately. In fact this has been done in [LR08, LSRS09] when all the tasks are under the control of a central authority, leading to a single objective optimization problem.

The first example above illustrates the usage of an explicit incentive mechanism in the pricing model: Users with DT tasks prefer to schedule their tasks when prices are low. The second example illustrates the main challenge addressed in this chapter: *How to provide the right incentives so that users sharing a common resource schedule their DT tasks during the periods of low utilization given a pricing system does not provide such incentives.* The later situation occurs often in practice, for example when using flat-rate payment models (commonly done by ISPs, enterprise, campus networks – for the last ones the flat-rate has value zero) there is no explicit incentive to postpone the execution of DT tasks. Even more, flat-rate pricing schemes are becoming the standard for end-users in telecommunications services [Odl01], precluding changes to the pricing model as the means to create the incentive mechanism.

Other challenges that arise in the design of a mechanism for this problem are:

- There should be a notion of fairness between the users of the system. Notions of fairness commonly used in networking, such as *max-min fairness* or *proportional fairness* are not appropriate for this problem as they refer to the instantaneous rates of the flows sharing the link, without regard to the user/application

they belong to. Briscoe [Bri07] points out (among many other drawbacks) that these per-flow metrics are easy to bypass simply by having multiple concurrent flows.

- The system should not rely on establishing valuations of individual tasks. A valuation is a quantitative measure of how much is a task worth for the user, and therefore is a commonly used parameter in many micro-economic mechanisms such as auctions. The problem as pointed out by Briscoe [Bri07] is that valuations are usually unknown or subjective, and even if known they would be impractical to communicate to the system.
- The system should not impose policies that violate the principle of “network neutrality” [Cro07]. Many ISPs have deployed systems to filter traffic based on the application. These mechanisms exclude legitimate uses of the network, are opposed to the openness nature of the Internet and could potentially be used for monopolizing certain services (for example an ISP blocking voice services to monopolize the voice market on his network). Many of these mechanisms depend on Deep Packet Inspection (DPI) techniques, which raises many privacy concerns and its long-term scalability is not clear as traffic grows and applications easily adapt to bypass them, for example using packet encryption and randomization of port numbers.
- The operation of the system should be as transparent as possible to the users. Ideally the user should not get involved in the operational aspects of the system, for example in doing any scheduling decision.

This chapter presents Trade & Cap, a marketplace-inspired mechanism that solves this problem and addresses all of the above mentioned challenges. This mechanism does not rely on real currency, but instead uses the qualities of different classes

of traffic as a *virtual currency* to enable the users to trade allocations in the marketplace. Section 5.1 describes the general scenario used by the marketplace mechanism, and although the examples refer to a shared network link as the contended resource, it should be pointed out that it is equally applicable to schedule other types of resources as well. Section 5.2 gives the model definition used by the different variants of the marketplace presented in this chapter. Section 5.3 presents gives the analytical description of the marketplace framework for the case of *non-elastic* or *fluid* applications, *i.e.* applications whose utility is continuous and strictly increasing as function of the total allocation. Next, Section 5.4 extends the previous results for the case of *inelastic* or *atomic* tasks. It is shown that under certain restrictions the system is guaranteed to reach a Nash Equilibrium and bounds on the Price of Anarchy are established. Sections 5.3 and 5.4 consider the operation of the marketplace when there is a single type of tasks, either fluid or atomic. In Section 5.5, it is shown that both schemes can be implemented together resulting in general version of the T&C marketplace. Section 5.6 describes a prototype implementation of the system build around the Hierarchical Token Bucket (HTB) network scheduler included in the Linux kernel. This implementation supports both fluid/atomic allocations and meets the neutrality requirement by placing all classification decisions on the client side of the service. Finally, Section 5.7 presents an experimental evaluation of the marketplace mechanism performed on real network traces, thus illustrating the benefits of the system under realistic conditions.

5.1 Application Scenario for Trade & Cap

Figure 5.1 illustrates the main application scenario used throughout this chapter. In this example there is a single bottleneck link and a set of users sharing it. This abstracts cases in which a set of subscribers to an ISP share the ISP's outgoing link, or

cases where users within an enterprise/campus network share a common uplink, etc. Commonly in these scenarios it is the case that there is no cooperation between the users, they simple execute their tasks (access web pages, checking e-mail, streaming media, downloads, backups) without regard/knowledge of other tasks being executed at the same time. In this scenario the sharing occurs at the edge of the network and the common access link (or a link on the path) is the bottleneck – *i.e.*, it is assumed that the core of the network is well provisioned and that its impact on QoS is small.

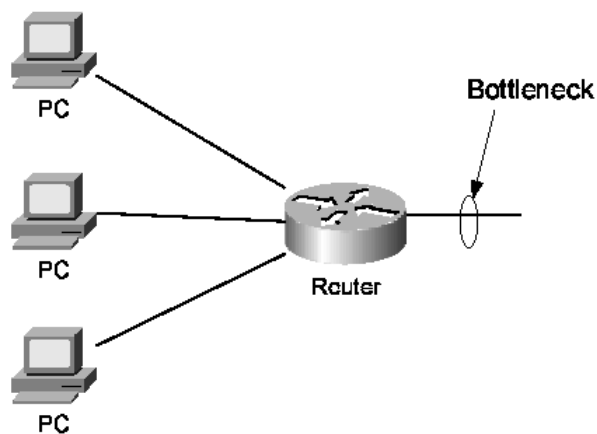


Figure 5.1: A BW sharing installation.

As a result of the lack of coordination between the users, the following negative effects are inevitable:

1. Delay sensitive applications (such as VoIP, video streaming, web browsing) suffer poor performance during periods of high utilization.
2. Operators incur high cost for the contended resource. This is frequently the case for ISPs and enterprise networks, whose cost is determined using the 95/5 rule, that sets the service charge based on the 95-percentile of the distribution of the 5-min traffic aggregates.
3. As the activity periods exhibit some diurnal patterns [MFPA09], the resource

goes over cycles of under/over-utilization. To ensure good performance during the periods high demand, the shared link must be over-provisioned, requiring significant investments in infrastructure.

5.2 Model Definitions

The system model assumes that the bandwidth consumption profile for each user follows some characteristic pattern – *e.g.*, the typical diurnal traffic pattern which is made up of the two constituents Interactive Traffic (IT) and Delay-Tolerant (DT). IT refers in general to the traffic due to applications that are delay sensitive (irrespective of their bandwidth requirement) such as VoIP, audio/video streaming, web-browsing, *ssh* sessions, etc. DT is due to applications that are not sensitive to delay, and have not stringent deadlines. It is assumed that DT applications benefit by having as much bandwidth as possible. Some examples of DT are replication/mirroring services, network backups, P2P file transfers, etc.

In the system’s model, time is quantized into time slots. A sequence of T consecutive time slots constitutes an allocation *epoch*. The IT pattern of each user is given by a vector (x_{i1}, \dots, x_{iT}) , where x_{ij} denotes the estimated IT demand for user i during time slot j . Note that the granularity of the time slots is inconsequential for the system.¹ The experimental evaluation presented later uses *5min* time slots, mirroring the granularity that is typically used for network monitoring/accounting purposes. The characterization of IT using the vector (\dots, x_{ij}, \dots) is meant to provide an estimate of future demand, and thus does not need to be exact. Furthermore, in order to provide a safety margin, the system enforces a minimum allocation per time slot, so that $x_{ij} \geq x_{min}$ for a configurable value of x_{min} . Similarly, the DT is described by a vector (\dots, w_{ij}, \dots) , whose components w_{ij} denote the allocation of

¹The system is concerned with long-term bandwidth management; it is not intended to operate at the time-scales of round-trip-times, but rather at time-scales that are orders of magnitude larger.

DT for user i at time slot j .

Users derive some utility through the satisfaction of their demands of both IT and DT. The utility derived by a user from the realization of its IT demand vector is assumed to be a fixed constant, whose precise value is not germane to the system. This is a desirable feature given the documented difficulties in establishing such a value, as discussed in this chapter's introduction. On the other hand, and in accordance with the notion of elasticity for the DT, the components w_{ij} are positive real variables and the larger the sum $\sum_j w_{ij}$, the greater the utility derived from DT applications. Section 5.3 describes a first version of the marketplace, where the users only trade allocations of DT components and their IT vector is constant throughout the allocation *epoch*. Section 5.4 extends the marketplace mechanism to allow users to trade allocations of IT components, but preserving the size of each component. For this reason this is referred to as trading of atomic components.

The trading mechanisms presented in this chapter implement a notion fairness between the users by assigning each user a number of *shares* which represent their rights for bandwidth reservation on the system. Thus, if B_i is the total shares (budget) of user i , and if all the users make uniform allocations over all time slots, then B_i is the amount of service received by user i . The budget does not need to be the same for all users, thus allowing for tiered service models, *e.g.* premium/standard/basic or prioritized service within an organization. The budget is thus used to enforce a notion of long-term economic fairness between the users, this in accordance to the notion of fairness suggested by Briscoe [Bri07]. A key feature of the system is that when the demand on the time slots is not uniform, the value of the shares will not be uniform, but will depend on the total demand per time slot. By doing this, the system will impose a supply/demand driven valuation of the time slots, thus providing the right incentive so that users prefer to allocate DT on slots with low demand.

This is accomplished by defining the cost (in shares) of a given allocation in a time slot as

$$c_{ij} = (x_{ij} + w_{ij}) \cdot f(U_j) \quad (5.1)$$

which indicates that c_{ij} shares are needed to allocate a total bw of $x_{ij} + w_{ij}$ in time slot j . The function $f()$ is a continuous, twice differentiable, convex, and strictly increasing function of the total slot utilization $U_j = \frac{1}{C} \sum_i (x_{ij} + w_{ij})$, being C the capacity of the resources. This function yields a bandwidth unit cost per-slot that is determined by the prevailing levels of supply and demand (“market”), whereby slots with low demand are cheap, and slots with high demand are expensive. The intuition of this product-form cost function is that it makes the time slot more expensive for users responsible for the largest fraction of the utilization. It is the best-strategy for the user to minimize its contribution w_{ij} on hot-spots, thus giving the incentive to load-balance the utilization of the link.

5.3 Demand-based Trading for Fluid Traffic

The initial version of the marketplace will assume that the IT components x_{ij} are constant and the users trade only the allocations of DT components w_{ij} . Since the goal of an individual (rational) user is to maximize its own utility, and since the utility of IT is constant and that of DT is positively correlated with the total volume of DT, it follows that the objective of user i would be to find the vector (\dots, w_{ij}, \dots) that maximizes its total DT allocation

$$\sum_{j=1}^T w_{ij} \quad (5.2)$$

subject to the budget and positivity constraints

$$\sum_{j=1}^T (x_{ij} + w_{ij}) \cdot f(U_j) \leq B_i \quad (5.3)$$

$$w_{ij} \geq 0 \quad \forall j = 1 \dots T \quad (5.4)$$

Solution of the per-user problem:

The per-user optimization problem can be solved using the Lagrange multipliers technique [Sun96] as follows:

Let $W_i = (\dots, w_{ij}, \dots)$ and define the benefit function for user i as

$$b_i(W_i) = \sum_{j=1}^T w_{ij}$$

Define also the constraint functions

$$h_{iq}(W_i) = \begin{cases} w_{iq} & q = 1, \dots, T \\ B_i - \sum_{j=1}^T (x_{ij} + w_{ij}) \cdot f(U_j) & q = T + 1 \end{cases}$$

Then the per-user optimization problem can be expressed as:

Maximize $b_i(W_i)$ subject to $W_i \in \mathcal{D} = \{W_i \in \mathbb{R}^T \mid h_{iq}(W_i) \geq 0, q = 1, \dots, T + 1\}$

Lemma 4. (Convexity of the feasible space \mathcal{D})

When the cost function $f()$ is a continuous, twice differentiable, convex, and strictly increasing function, the set of feasible solutions $\{W_i \in \mathcal{D}\}$ is convex.

Proof. Each one of the functions $h_{iq}(W_i)$ defines a convex set S_q as follows:

For $q = 1, \dots, T$, the functions $h_{iq}(W_i) = w_{iq}$ are linear, therefore concave, and the subgraph of $h_{iq}(W_i)$, S_q , is convex.²

To establish the concavity of $h_{i,T+1}(W_i)$ observe that its Jacobian is the $1 \times T$ vector

$$\begin{aligned} Dh_{i,T+1}(W_i) &= \left[\frac{\partial h_{i,T+1}(W_i)}{\partial w_{ij}}, \dots \right] \\ &= \left[-f(U_j) - (x_{ij} + w_{ij}) \frac{\partial f(U_j)}{\partial w_{ij}}, \dots \right] \end{aligned}$$

²Refer to [Sun96] §7.1.1

And its Hessian matrix is the $T \times T$ matrix

$$\begin{aligned} D^2 h_{i,T+1}(W_i) &= \left[\frac{\partial^2 h_{i,T+1}(W_i)}{\partial w_{ij} \partial w_{ik}}, \dots \right] \\ &= \left[-\frac{\partial f(U_j)}{\partial w_{ik}} - I_{(j=k)} \frac{\partial f(U_j)}{\partial w_{ij}} - (x_{ij} + w_{ij}) \frac{\partial^2 f(U_j)}{\partial w_{ij} \partial w_{ik}} \right] \end{aligned}$$

where $I_{(j=k)}$ is the indicator function that returns 1 if $j = k$ and 0 otherwise. Observing that $\frac{\partial f(U_j)}{\partial w_{ik}} = 0$ when $j \neq k$, and $\frac{\partial^2 f(U_j)}{\partial w_{ij} \partial w_{ik}} = 0$ when $j \neq k$, then the Hessian is a diagonal matrix whose diagonal elements are:

$$\alpha_{jj} = -2 \frac{\partial f(U_j)}{\partial w_{ij}} - (x_{ij} + w_{ij}) \frac{\partial^2 f(U_j)}{\partial w_{ij}^2}$$

Also, for any arbitrary vector z different from zero,

$$\begin{aligned} z^T [\alpha_{ij}] z &= [z_j \alpha_{jj}] z \\ &= \sum_j z_j^2 \alpha_{jj} \\ &< 0 \end{aligned} \tag{5.5}$$

where the last step uses the fact that $\alpha_{jj} < 0$ because of the assumption that $f(\cdot)$ is twice differentiable, convex (therefore $\frac{\partial^2 f(U_j)}{\partial w_{ij}^2} > 0$) and strictly increasing (so that $\frac{\partial f(U_j)}{\partial w_{ij}} > 0$). Hence, the Hessian $D^2 h_{i,T+1}(W_i)$ is negative definite and $h_{i,T+1}(W_i)$ is concave (See Theorem 7.10 [Sun96]). In consequence, the subgraph S_{T+1} of $h_{i,T+1}(W_i)$ is a convex set, and the set

$$\mathcal{D} = \cap_{q=1}^{T+1} S_q$$

is also convex (See Theorem 1.37 [Sun96]). \square

Having established the convexity of the feasible space, then the existence of the maximum and the way to determine that maximum are established by the following theorem:

Theorem 8. (Optimal solution for the per-user problem)

Given: 1) a cost function $f(\cdot)$ continuous, twice differentiable, convex and strictly increasing, and 2) a fixed allocation x_{ij} whose cost is below the budget B_i , then there exists a $W_i^ \in \mathcal{D}$ that maximizes $b_i(W_i^*)$ and a vector λ^* that satisfies the Kuhn and*

Tucker conditions for this problem.

Proof. The cost of the fixed allocations is below the budget, *i.e.*

$$\sum_{j=1}^T x_{ij} f(\mu_j) < B_i$$

where μ_j is the slot utilization when all the $w_{ij} = 0$, *i.e.* $\mu_j = U_j - w_{ij}/C$. Given that the cost function is continuous over all the variables w_{ij} , there exists a vector $\bar{W}_i = (\bar{w}_{ij}, \dots)$ such that

$$h_{iq}(\bar{W}_i) > 0, \quad q = 1, \dots, T+1$$

(this is known as *Slater's condition*). Also, given that the feasible space \mathcal{D} is a bounded convex set, it follows that at least one of its elements maximizes $b_i(W_i)$. The objective function $b_i(W_i)$ is a concave continuous function and the constraint functions $h_{iq}(W_i)$ are also concave (which was established in the proof of Lemma 4). As a result all the conditions of the theorem of Khun and Tucker under convexity (Theorem 7.16 in [Sun96]) are satisfied, and the existence of a maximum W_i^* implies there is an associated vector λ^* that satisfies the Kuhn-Tucker first order conditions, namely

$$\begin{aligned} \text{[KT - 1]} \quad & Db_i(W_i^*) + \sum_{q=1}^{T+1} \lambda_q^* Dh_{iq}(W_i^*) = 0 \\ \text{[KT - 2]} \quad & \lambda^* \geq 0, \sum_{q=1}^{T+1} \lambda_q^* h_{iq}(W_i^*) = 0 \end{aligned}$$

□

The previous theorem constitutes a solution method for determining the per-user best-response in a T&C marketplace of fluid allocations. The exact solution depends on the assignment of the cost function, the following example illustrates the case when $f()$ is the identity function.

Example 1. *Solution to the user's optimization problem when the cost function is $f(U_j) = U_j$.*

The Laplacian associated with the user's optimization problem is

$$L(W_i, \lambda) = \sum_{p=1}^T w_{ip} + \sum_{p=1}^T \lambda_p w_{ip} + \lambda_{T+1} \left(B_i - \sum_{p=1}^T (x_{ip} + w_{ip}) U_p \right) \quad (5.6)$$

The condition KT-1 establishes that at the point W_i where $b_i(W_i)$ is maximum, there is a vector λ such that

$$\frac{\partial L(W_i, \lambda)}{\partial w_{ij}} = 1 + \lambda_j - \lambda_{T+1} \left(U_j + (x_{ij} + w_{ij}) \frac{1}{C} \right) = 0 \quad \text{for } j = 1, \dots, T \quad (5.7)$$

(using the fact that $U_j = \frac{1}{C} \sum_i (x_{ij} + w_{ij})$, therefore $\frac{\partial U_j}{\partial w_{ij}} = \frac{1}{C}$).

By condition KT-2, the sum of the constraint functions at the optimal point, each term weighted by its respective Lagrange multiplier is zero, and the Lagrange multipliers are greater or equal to zero. In particular the constraint function $h_{i,T+1}(W_i)$ refers to the budget and this constraint is always satisfied with equality under the conditions of the T&C market, otherwise any left over budget can always be used to increase one or more of the w_{ij} and the point W_i would not correspond to the maximum. Hence, after removing this term, condition KT-2 simplifies to:

$$\lambda \geq 0, \quad \sum_{j=1}^T \lambda_j w_{ij} = 0$$

Consequently and as the values $w_{ij} \geq 0$ and $\lambda_j \geq 0$, it must be the case that for every j either $\lambda_j = 0$ or $w_{ij} = 0$. Solving for w_{ij} from eq. (5.7) for the cases where $\lambda_j = 0$

$$w_{ij} = \frac{C - \lambda_{T+1}(\mu_j C + x_{ij})}{2\lambda_{T+1}} \quad (5.8)$$

Replacing the values of w_{ij} in the constraint $h_{i,T+1}(W_i) = 0$, and reorganizing terms gives

$$\begin{aligned} B_i - \sum_p (x_{ip} + w_{ip})(\mu_p + w_{ip}/C) &= 0 \\ CB_i - \sum_{p|\lambda_p=0} (w_{ip}(x_{ip} + \mu_p C) + w_{ip}^2) - C \sum_{p|\lambda_p \neq 0} x_{ip} \mu_p &= 0 \end{aligned} \quad (5.9)$$

By substitution of eq. (5.8) into (5.9), and by applying several simplification steps:

$$\begin{aligned}
0 &= \lambda_{T+1}^2 \left(\sum_{p|\lambda_p=0} (x_{ip} + \mu_p C)^2 + 4CB_i - 4C \sum_p x_{ip} \mu_p \right) - \sum_{p|\lambda_p=0} C^2 \\
\lambda_{T+1} &= \sqrt{\frac{\sum_{p|\lambda_p=0} C^2}{\sum_{p|\lambda_p=0} (x_{ip} + \mu_p C)^2 + 4CB_i - 4C \sum_p x_{ip} \mu_p}} \quad (5.10)
\end{aligned}$$

The only problem left to solve this equation is determining the time slots p such that the corresponding Lagrange multipliers are $\lambda_p = 0$. An iterative solution procedure was devised as follows: 1) Remove all the constraints $h_{iq}(W_i)$ for $q = 1, \dots, T$. This is equivalent to lifting the restriction of the values w_{ij} being positive. 2) Solve the problem without these constraints. If all the values are positive, it is also a solution to the problem with the positivity constraint. If not, mark the time slots with negative w_{ij} as slots with $\lambda_p \neq 0$. Then solve again using eq. (5.10). Observe that the exact value of λ_p is not required, only the knowledge of them being zero or not.

Theorem 9. *The global optimum of the user's optimization problem is unique*

Proof. Suppose it is not. Let X and Y be two distinct global maximizers of $b_i()$. Let $Z = \alpha X + (1 - \alpha)Y$ for $\alpha \in (0, 1)$. By the convexity of \mathcal{D} , it is always the case that $Z \in \mathcal{D}$. By the linearity of $b_i()$

$$\begin{aligned}
b_i(Z) &= \alpha b_i(X) + (1 - \alpha)b_i(Y) \\
&= b_i(X)
\end{aligned}$$

the second step because under the conditions of Theorem 8 any maximizer of $b_i()$ is a global maximizer. This means that all the points z in the hyperline segment defined by X, Y are also global maximizers. The constraint $h_{i,T+1}()$ is strictly concave (as shown in Lemma 4), therefore

$$h_{i,T+1}(Z) > \alpha h_{i,T+1}(X) + (1 - \alpha)h_{i,T+1}(Y)$$

Remember that $h_{i,T+1}(Z)$ is the left-over budget at point Z . As discussed before,

at the optimal allocations X, Y all the budget has been used, therefore $h_{i,T+1}(X) = h_{i,T+1}(Y) = 0$. Having a left-over budget greater than zero at Z means that it is possible to increase the allocations in some time slots, which would give a greater value of $b_i()$ thus contradicting the optimality at Z . \square

The previous theorems related to the actions of a single user assuming the allocations of the other users remain fixed. In practice, if a user changes its allocations, the unit prices per slot (dictated by $f(U_j)$) change and the optimal solution for other players also changes. It is then possible for the users of the fluid T&C marketplace to engage in a game where each user changes its strategy (its allocation vector W_i) in response to other users actions.

Theorem 10. *If the set of mutually feasible allocations is not empty, the T&C marketplace has a Nash Equilibrium.*

Proof. Define the global per-slot allocations as $w_{gj} = \sum_{i=1}^n w_{ij}$. Also define the global fluid optimization problem as follows:

Maximize

$$\sum_{j=1}^T w_{gj} \quad (5.11)$$

subject to the constraints

$$\sum_{j=1}^T (x_{ij} + w_{ij}) \cdot f(U_j) \leq B_i \quad \text{for } i = 1, \dots, n \quad (5.12)$$

$$w_{ij} \geq 0 \quad \text{for } i = 1, \dots, n, \quad \text{and } j = 1 \dots T \quad (5.13)$$

The laplacian of this problem is:

$$L(W_g, \lambda) = \sum_{p=1}^T w_{gp} + \sum_{p=1}^T \lambda_p w_{gp} + \sum_{i=1}^n \lambda_{T+1,i} (B_i - c_i) \quad (5.14)$$

Observe that this Laplacian is the sum of the per-user Laplacians (eq. 5.6), and the condition [KT-1] is satisfied if and only if the per-user [KT-1] is satisfied. In consequence, the Lagrange multipliers that solve the per-user maximization problem (eq. 5.10) are also the ones that solve the global maximization problem. In other

words, the vector W_g^* that maximizes (5.11) is obtained when the corresponding per-user allocations W_i^* are optimal.

As the set of feasible allocations is not empty, one of its members maximizes (5.11) and this point corresponds to an per-user optimal as well, therefore it is a NE. \square

An illustrative example:

Figure 5.2 illustrates the equilibrium solution for a case with two users that have interactive traffic profiles that overlap during some periods (the solid line). The dashed line is the total allocated bandwidth per user for a budget of $256kbps$ per time slot. The indicated time slots T_1 and T_2 illustrate cases where the allocation of DT of one user reduces to give opportunity to handle a peak of IT from the other user. Notice that a conventional liquid filling algorithm would not achieve this result.

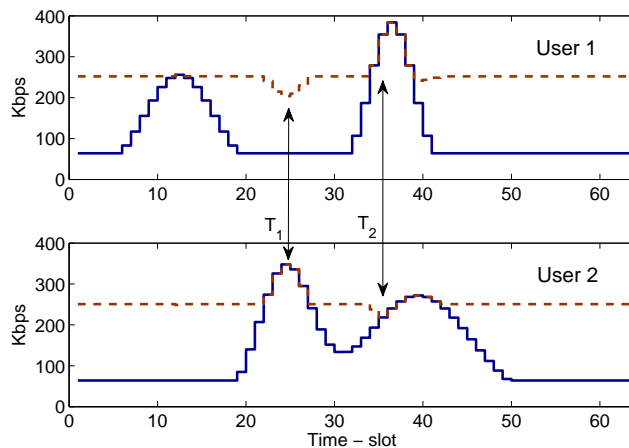


Figure 5.2: Example with two users

On the Notion of Fairness: The scheme presented in this section uses a notion of fairness different from the classical definitions of *max-min* fairness and *proportional fairness*. In particular, these definitions compare the instantaneous rates of various flows sharing the link. They do not take into account the number of flows per user or when these flows are active. The T&C mechanism uses a different notion: All of users of the shared resource pay exactly the same unit-price during any time slot.

Users are free to allocate any amount and may choose where in time they want to make their reservations. The unit price itself (computed as $f(U_j)$) is variable and it is this variability that introduces the incentive for users with DT tasks to move them away from the periods of high demand.

5.4 Demand-based Trading for Atomic Tasks

The previous section considered the problem when traffic can be handled as a fluid: any positive real-valued allocation is valid. An alternative problem is when the tasks are atomic, *i.e.* the task has to be executed entirely or not at all. Partial execution has no value for the user and any partial work is only wasting system resources. In this case the T&C marketplace can also provide the right incentives for users to schedule their DT tasks during periods of low utilization, and at the same time minimizing the pressure on the time slots of high demand. To maintain the connection with the same practical example of the previous section, the atomic tasks will be referred to as the IT components of the network activity. Interactive tasks usually have a finite size and a deadline. For example when streaming a video, each frame has a finite size and must be received within some limited time in order to be ready at playback time.

To describe the possibility of re-allocating atomic tasks, it is necessary to introduce a few extensions to the notation used in the previous sections: Let each agent i represent its IT demand as a vector of requested bandwidth allocations: $T_i = (t_{i1}, \dots, t_{il_i})$. An assignment of an agent's demand is a mapping that pins each one of the components of the vector to a different time slot. A set of such assignments (one per agent) comprises a potential configuration or *schedule* of IT utilization at the shared link. Let $k = m_i(j)$ be the time slot assigned to the j^{th} component of user i 's request vector. Denote by x_{ik} the actual allocation for player i in time slot k , *i.e.*

$x_{ik} = t_{i,m_i(j)}$. The x_{ik} notation implicitly represents the mapping $m_i()$, noting that for time slots that are not used in the mapping, it is assumed that $x_{ik} = 0$. The x_{ik} values thus defined are the same “fixed” components as used in the previous section.

The T&C marketplace is responsible for facilitating the trading process between the users, providing the incentives and enforcing the outcome of the market. The interactions of the users in the market are modeled as a pure-strategies non-cooperative game where each user chooses an action base solely on its own selfish and rational interest.³ The strategy space S_i^* for agent i is the set of permutations of its request vector T_i . As such, the strategy space is finite with cardinality $|S_i^*| = P_{l_i}^T$. The game’s strategy space is the Cartesian product of the strategy spaces of all players: $\mathcal{S} = \times S_i$. For simplicity of exposition, the initial presentation assumes all the points in the strategy space are valid choices, and later on these results will be extended to consider the existence of constraints in each player’s strategy space.

Theorem 11. *The pure strategies game in which users adopt better/best responses to allocate atomic units of traffic in per-user, mutually-exclusive time slots converges to a NE when the per-slot cost function is the identity function.*

Proof. Define the following function:

$$\Phi = \sum_{i=1}^n \sum_{j=1}^T c_{ij} = C \sum_{j=1}^T U_j^2$$

where n is the number of users and T the number of time slots.

Being the cost function $f(U_j) = U_j$, when a player makes a cost-reducing move, $\Delta c_i < 0$,

$$\sum_j (x'_{ij} U'_j - x_{ij} U_j) < 0 \quad (5.15)$$

where x'_{ij} , U'_j denote the user allocation and the total utilization after the execution

³It should be pointed out that although the rationality assumption has been questioned in many socio-economic settings, the resource management scheme considered in this thesis assumes that the trading is done by utility maximizing software agents, thus making this consideration unnecessary.

of the move.

Notice that for any other player $k \neq i$, its utilization of interval j does not change, but the change in the total utilization affects its cost as follows

$$\Delta c_k = \sum_j x_{kj}(U'_j - U_j)$$

Adding the changes of the players other than i

$$\begin{aligned} \sum_{k \neq i} \Delta c_k &= \sum_{k \neq i} \left(\sum_j x_{kj}(U'_j - U_j) \right) \\ &= \sum_j \left((U'_j - U_j) \sum_{k \neq i} x_{kj} \right) \\ &= \frac{1}{C} \sum_j \left((x'_{ij} - x_{ij}) \sum_{k \neq i} x_{kj} \right) \end{aligned} \quad (5.16)$$

where the last step uses the fact that $U'_j - U_j = \frac{1}{C}(x'_{ij} - x_{ij})$ because players other than i did not change their allocations. The atomicity of the components implies that the vector (x'_{ij}, \dots) is a permutation of the elements of the vector (x_{ij}, \dots) , therefore $\sum_j x'^2_{ij} = \sum_j x^2_{ij}$. Then expression (5.15) can be re-arranged as follows

$$\sum_j (x'_{ij}U'_j - x_{ij}U_j) = \sum_j \left((x'_{ij} - x_{ij}) \sum_{k \neq i} x_{kj} \right) < 0$$

The last expression and the fact that $C > 0$ allows to conclude that expression (5.16) is also negative, hence $\Delta c_i + \sum_{k \neq i} \Delta c_k = \Delta \Phi < 0$. *i.e.* Φ is strictly decreasing and knowing that it is lower-bounded by some constant greater than zero, this allows to conclude Φ is a potential for the game. \square

Convergence with constraints:

As alluded before, it may be the case that an agent may have additional constraints that limit its strategy space – *e.g.*, a 2-hour-long IT fixed bandwidth allocation must be assigned in consecutive time slots, and be scheduled to start between 6pm and 8pm. Such constraints are easily captured by defining the player's strategy space

as a subset of $S_i \subseteq S_i^*$. This particular case was included in both the prototype implementation and the experimental evaluation of the T&C mechanism. For this purpose, the IT tasks are defined as collections of non-overlapping *sessions*, where each session represents a task larger than the time slot quantum, but must be allocated atomically – *i.e.* its component subtasks must be allocated to consecutive time slots. The proposed implementation also allows specifying a *slack* indicating the number of time slots the tasks may be shifted back or forward on the schedule.

Other types of constraints important in practice are: (1) *Capacity constraints* to ensure that the shared link capacity is never exceeded by the aggregate allocation – $\forall j : \sum_{i=1}^n x_{ij} \leq C$, and (2) *Budget constraints* to ensure that no agent is able to reserve resources beyond his *fair share*, which is upper-bounded by the agent’s allowance – $\forall i : \sum_{j=1}^T x_{ij} U_j \leq B_i$.

Notice that both sets of constraints correspond to the elimination of infeasible points in the strategy space \mathcal{S} . This removal can be easily accomplished by setting to ∞ the cost per player for those points.

Theorem 12. (Convergence to NE under constraints) *Given a pure strategies game, such that each player’s action space is finite, and that better/best response dynamics converge to a NE, then after adding constraints to the action space of one or more players, the better/best response dynamics still converge, given that there exists feasible configurations after the addition of the constraints.*

Proof. Consider the following directed graph $G = \langle V, E \rangle$: There is a vertex $v_j \in V$ for every possible point in the strategy space $v_j = (a_{1j_1}, \dots, a_{nj_n})$, where a_{ij} denotes the j^{th} action of player i . There is an edge $e_{pq} \in E$ for any valid transition⁴ on the strategy space, *i.e.* the cost associated with player i at vertex p is larger than the cost at vertex q : $c_p(i) > c_q(i)$ and $a_{-i,p} = a_{-i,q}$, meaning that the actions of all players other than i are the same in p and q . Let us call G the transition graph of the game. Then, if the better/best-response dynamics always converge to a NE in the unconstrained case, G is a DAG. Any path (sequence of actions) the players

⁴Observe that the set of edges is not limited to best-responses, but includes any feasible move

traverse when following their rational-selfish goal will always reach a vertex with no outgoing edges corresponding to a NE (of possibly many) of the game. The addition of constraints to the players actions, corresponds to removing unfeasible vertices from V as well as the edges coming into or out of these vertices. Let G' be the residual transition graph after removing unfeasible vertices and edges. Suppose the better/best-response dynamics with constraints do not always converge to a NE. Then, there exists at least one cycle in the residual transition graph G' . Being G' a subgraph of G this implies the same cycle must exist in the original graph G , contradicting the fact that G is a DAG. \square

Figure 5-3 illustrates the construction used in the proof. In (a) it shows the DAG corresponding to the transitions of some hypothetical game, where states v_6 and v_8 are the NE. In (b) v_4 and v_6 have been removed with their respective edges because they are unfeasible. The NE in the residual graph are v_3 and v_8 . Notice that the set of NE vertices after the addition of constraints need not to be the same as those of the unconstrained game. In particular, feasible vertices that were not a NE will become a NE if all their outgoing edges are removed.

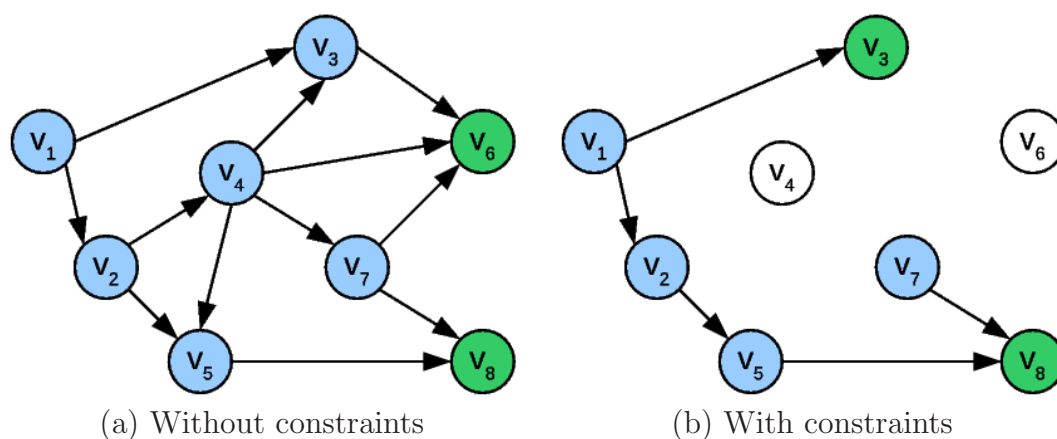


Figure 5-3: Transition graphs for a pure strategies game.

Price of Anarchy (PoA):

An important consideration when considering equilibria of non-cooperative games is the Price of Anarchy – the ratio of the social cost at the worst-case equilibrium com-

pared to the optimal cost. In the Atomic Trading case, the social cost (understood as the metric the system wants to optimize) is the maximum resource utilization.

Theorem 13. (Price of Anarchy for Atomic Trading with no constraints) *The PoA of the Atomic Trading game with no constraints is 2.*

Proof. Consider the interval with minimum usage⁵ u_{min} . Then for all intervals j ,

$$u_{min} \leq u_j$$

Assume the game is at a NE. Let x, y be the size of tasks of user i on intervals min and j respectively. Then by the equilibrium condition it must be the case that for all intervals j

$$\begin{aligned} xu_{min} + yu_j &\leq y(u_{min} - x + y) + x(u_j - y + x) \\ &\leq yu_{min} + xu_j + (x - y)^2 \\ (x - y)(u_{min} - u_j) &\leq (x - y)^2 \end{aligned} \tag{5.17}$$

If $x \geq y$, eq. (5.17) is always satisfied as the left hand side is always negative or zero and the right hand side is always zero or positive. If $x < y$, eq. (5.17) can be simplified as

$$u_j - u_{min} \leq y - x \tag{5.18}$$

in other words, eq. (5.18) gives an upper bound for the difference between any interval and the interval with least utilization. In particular, it holds for the interval with highest utilization:

$$u_{max} - u_{min} \leq y - x \tag{5.19}$$

Let T be the total number of slots, and $M = \sum_i \sum_j x_{ij}$ be the total mass. Then, the following relation always holds:

$$u_{min}(T - 1) + u_{max} \leq M \tag{5.20}$$

A perfectly even distribution of the mass on all the slots would give an usage of M/T per slot, and if t_{max} is the maximum size of any subtask, then a lower bound for the

⁵Usage refers to the total allocation previous to normalizing by the capacity, *i.e.* $U_j = u_j/C$.

optimal maximum usage can be established as follows:

$$\max\{t_{max}, M/T\} \leq OPT$$

so, it must be the case that $M \leq T \cdot OPT$ and $t_{max} \leq OPT$. Replacing the first condition on (5.20) gives

$$u_{min}(T - 1) + u_{max} \leq T \cdot OPT \quad (5.21)$$

Also, as t_{max} is the upper bound of the task size, then $y - x \leq t_{max}$, therefore from the second condition in (5.19)

$$u_{max} - u_{min} \leq OPT$$

or

$$u_{max} - OPT \leq u_{min}$$

this is a lower bound on u_{min} , which after replacing in (5.21) gives

$$(u_{max} - OPT)(T - 1) + u_{max} \leq T \cdot OPT$$

and after some simplification

$$u_{max} \leq \left(\frac{2T - 1}{T}\right) OPT < 2 \cdot OPT$$

i.e. the PoA is 2. □

This bound is tight, as illustrated by the example in Fig. 5.4. In this example, all the tasks are singletons (chains of length=1). Part (a) illustrates a NE where the maximum load is $2t_{max}$. Part (b) illustrates an optimal allocation where the maximum load is $t_{max} + \varepsilon$. Setting $\varepsilon = t_{max}/T$, the ratio between the load at NE and the optimal solution approaches 2 as T becomes very large.

As previously stated, practical applications may impose constraints on the set of valid allocations (strategies) for the user. In particular, one of those constraints is the ordering constraint, which states that for all pairs t_{ij}, t_{ik} such that $j < k$ it must be

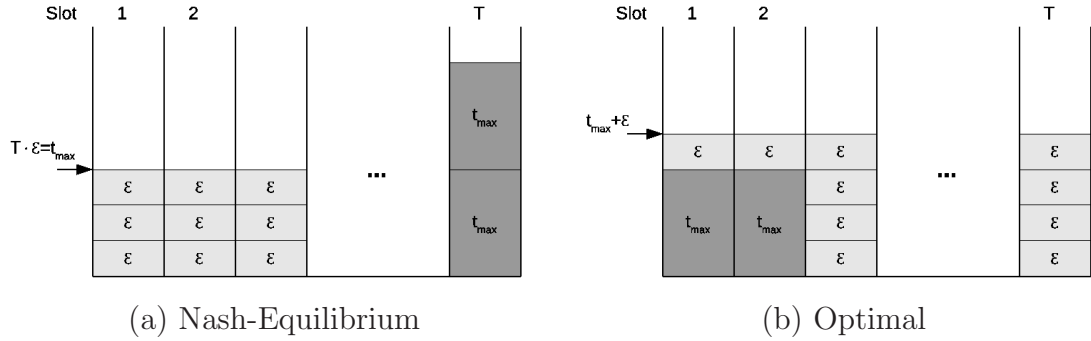


Figure 5-4: Tight example of PoA for unconstrained game

the case that the mappings $m(j) < m(k)$, *i.e.* the relative ordering of the subtasks is preserved. A typical example is when tasks have functional dependencies – a subtasks cannot be executed until its antecessor is completed. As pointed out previously, the NE when there are constraints may not be the same, and the unconstrained PoA no longer holds. The following theorem states the PoA of the game with ordering constraints.

Theorem 14. (Price of Anarchy for Atomic Trading with ordering constraints)
When user sessions are described as finite ordered sequences of atomic tasks, the PoA on the per-slot load is n .

Proof. A loose bound on the PoA for the trading game is trivial: Given a maximum allocation per user, X_{max} , it may be the case that all the n users have an equally-large demand, and there exists a NE where these demands coincide in the same time slot. On the other hand, there is always going to be a slot with utilization of at least X_{max} , therefore this is a lower-bound on the slot usage. This gives an initial set of bounds

$$X_{max} \leq \max\{u_j\} \leq nX_{max}$$

These loose bounds immediately imply that

$$PoA = \frac{\max_{\text{worst-case}}\{u_j\}}{\max_{\text{optimal}}\{u_j\}} \leq n$$

To show that this bound is tight, Fig. 5-5 shows an instance that realizes it. In this example there are n players, each one having a session of length $n + kn$, and

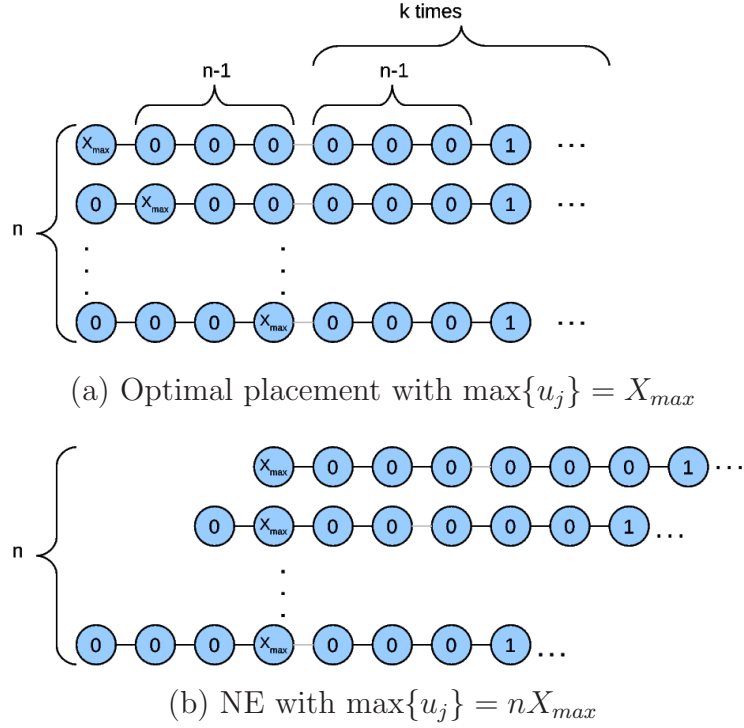


Figure 5-5: Tight example for the PoA of the Bandwidth Trading Game

the total number of time slots is $n + kn + n - 1 = n(k + 2) - 1$. Fig. 5-5a shows the optimal allocation which yields an $\max\{u_j\} = X_{max}$, and part (b) shows a NE whose $\max\{u_j\} = nX_{max}$. Part (b) is a NE because any unilateral deviation by any player, gives an higher cost. In fact the player cost at NE is

$$c_i = \frac{1}{C} \sum_j x_{ij} u_j = \frac{nX_{max}^2 + k}{C}$$

And the cost for a player if he moves any integral number of positions (within the allowed time slots) is

$$c'_i = \frac{1}{C} \sum_j x_{ij} u_j = \frac{X_{max}^2 + 2k}{C}$$

and $c'_i > c_i$ whenever $k \geq (n - 1)X_{max}^2$. □

The Price of Anarchy is a worst-case metric and for this particular problem realizing it requires a carefully crafted problem instance. To explore its behavior when the sets of tasks have randomly distributed sizes, a series of simulations were

conducted, according the following procedure:

1. Create a problem instance whose optimal allocation is known. The load-balancing problem itself is NP-Complete⁶. On the other hand, constructing an instance with a known optimal solution is simple: Take the slots, assume they are all equally filled say with 1 unit. Split the content of each slot in several fractions and then take sequences of elements from different slots to be the tasks of the players. Finally, shuffle around the tasks of the players and this gives a problem instance.
2. For different numbers of players (this defines the game size) and of time slots we create multiple problem instances. For this experiment 100 instances were evaluated per game size.
3. Run the game by letting the players take turns and play their best response until the game reaches a NE. Take the maximum among all the instances of the same size, and then compute the ratio with respect to the known optimum. This gives the empirical ratio of the worst-case to the optimal.

The results of these simulations are shown in Figure 5-6, with 5 slots (a) and 10 slots (b). Empirically, the PoA for the atomic trading is generally below 2, and tends to be insignificant as the number of players (size of the game) increases, which bodes well for T&C’s targeted applications.

⁶It is easy to see this by reduction from the 2-PARTITION [GJ79] problem. If an algorithm could solve the load-balancing problem in polynomial time, then it would solve the partition problem in polynomial time as well. Simply run the load-balancing algorithm with two slots using the same inputs of the partition problem. If the sum of the elements on each slot is equal, the answer to the PARTITION problem is “yes”, otherwise is “no”

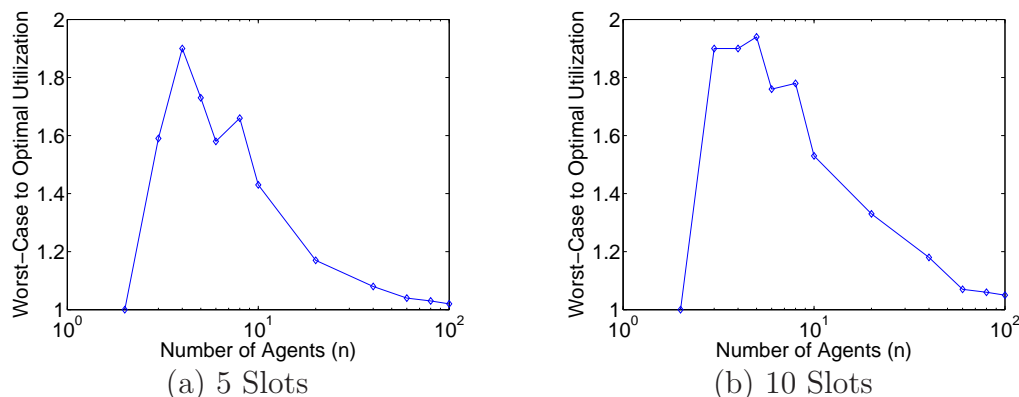


Figure 5.6: Empirical Price-of-Anarchy based on synthetic worksets

5.5 T&C: Combining Atomic and Fluid Tasks

The previous two sections consider the trading mechanism when there is a single class of jobs in the system, either fluid or atomic. There are many cases in computing systems where both kinds of tasks must be handled simultaneously. For example an end user may be viewing a streamed video and doing some downloads. The video stream needs to deliver a frame every frame-period, otherwise the playback quality will be very poor.⁷ On the other hand, the downloads do not suffer by changes in the rate, the perception of quality depends only on the total download time which is minimized when the download increases its share of the capacity over time.

5.5.1 Problem Formulation

Each user in the system has a profile of atomic tasks given by the vector $T_i = (t_{i1}, \dots, t_{i_i})$. The allocation of the tasks in the system has an associated constant utility which is larger than the utility obtained from the fluid tasks. The utility derived from the fluid allocation is strictly increasing with respect to the total fluid reservation, but the time at which the reservation is made is irrelevant. The user's

⁷For the sake of simplicity this example ignores the buffering process normally used when streaming video, but this does not change the fundamental issue: n-frames must be delivered each n-periods or the buffer will underflow and the playback will freeze.

objective is to maximize its total utility subject to a budget constraint in the cost of the reservations made, and possibly some constraints the assignment of atomic tasks to time slots. The mapping of the atomic tasks into the corresponding time slot is defined as $m : \{1, \dots, l_i\} \rightarrow \{1, \dots, T\}$, so that $k = m(j)$ is the slot assigned to task t_{ij} , being T the number of slots in the system. For notational simplicity, $x_{ik} = t_{ij}$ when task j is mapped into slot k , or zero if no task is mapped to slot k . $X_i = (x_{ik}, \dots)$ is the vector of allocations of atomic tasks. The fluid allocations are indicated by the vector $W_i = (w_{ij}, \dots)$.

It is assumed that the allocation of atomic tasks is always feasible, meaning that at each user is guaranteed the utility associated its atomic tasks. In this case, and being the utility of fluid tasks strictly increasing with respect to the total fluid allocation, the user's objective is to find the mapping m and the vector of fluid allocations W_i that maximizes

$$\sum_{j=1}^T w_{ij}$$

subject to the budget constraint⁸

$$\sum_{k=1}^T (x_{ik} + w_{ik})U_k \leq B_i \quad (5.22)$$

The combination of the discrete and the real components in the solution of this problem makes its solution more challenging. The following theorem allows separating the solution of the atomic allocation and the fluid allocation.

Theorem 15. *Given the identity cost function $f(U_j) = U_j$, in the case where there is fluid allocated in all slots ($\lambda_p = 0$, $p = 1, \dots, T$), the maximum user fluid allocation occurs when the cost of its atomic tasks X_i^* is minimal.*

Proof. Let $X_i \neq X_i^*$ be an allocation of atomic tasks that does not give the minimum cost. Let W_i be the vector of fluid allocations corresponding to the solution of the

⁸here the cost function is restricted to the case of $f(U_j) = U_j$ as the results of Section 5.4 have been proven only for this case.

per-user optimal when the atomic allocations are X_i . Then, from eq. (5.8) and after some algebraic manipulation we have

$$\begin{aligned}\sum_j w_{ij} &= \frac{TC}{2\lambda_{T+1}} - \frac{1}{2} \sum_j (\mu_j^C + x_{ij}) \\ &= \frac{TC}{2\lambda_{T+1}} - \sum_j x_{ij}\end{aligned}$$

Assume (for the sake of contradiction) that $\sum_j w_{ij} > \sum_j w_{ij}^*$. Then, by atomicity, $\sum_j x_{ij} = \sum_j x_{ij}^*$, therefore this implies

$$\begin{aligned}\frac{TC}{2\lambda_{T+1}} &> \frac{TC}{2\lambda_{T+1}^*} \\ \lambda_{T+1}^* &> \lambda_{T+1}\end{aligned}\tag{5.23}$$

However, from eq. 5.10 for this case (all $\lambda_p = 0, p = 1, \dots, T$) the value

$$\lambda_{T+1}^2 = \frac{TC}{4B}$$

is independent of the particular X , thus $\lambda_{T+1} = \lambda_{T+1}^*$ contradicting 5.23. \square

The previous result would give a direct methodology for the optimal T&C marketplace if the optimal allocation of the atomic components was easy to find. Unfortunately, this problem is an NP-complete problem, as stated in the following theorem.

Theorem 16. *Finding the optimal allocation of atomic tasks in the T&C marketplace is NP-complete*

Proof. By reduction to the 2-PARTITION problem. Take an instance of 2-PARTITION defined by the set $\{x_i, \dots\}$. Let each x_i be an independent tasks (one per user). Let the total budget be $B = 2 \left(\frac{1}{2} \sum_i x_i\right)^2$, and set the number of slots to 2. If there is a 2-partition of the set, the unit price per slot is $p = \frac{1}{2} \sum_i x_i$, so give each user a budget $B_i = p * x_i$. If the T&C marketplace could find the optimal solution for this problem in polynomial time, it would solve any instance of 2-PARTITION in polynomial time. \square

A Practical Approach: The lack of a polynomial-time solution for atomic case, which combined with the fluid would give a globally optimal allocation and a NE equilibrium for all the users, the practical approach used by the T&C marketplace is to let the user bid until the marketplace reaches a NE. Being a local-optimal, the NE solution is then used to compute the fluid allocations for all the users.

5.6 Prototype Implementation

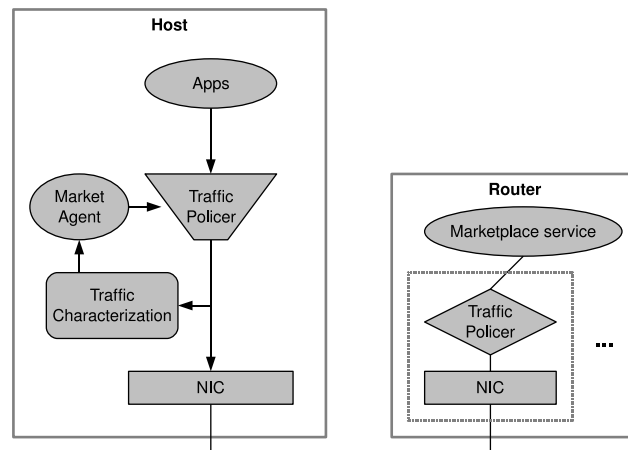


Figure 5-7: Overall T&C Architecture

The general architecture of the system is illustrated in Figure 5-7. Its main components are:

1. **Traffic Characterization:** Uses a statistical characterization technique to create a profile of the user's IT demand.
2. **Marketplace:** Divided into user-side and router-side agents. The user-side takes the user's profile and submits its request to the server side-agent. The server-side agent accepts the bids from the user agents and reports back the corresponding costs. This iterative process stops when the marketplace reaches a NE, which settles the allocations for an *epoch*.

3. **Traffic Policing:** The outcome of the marketplace is enforced by the policing agents. A policing agent on the router enforces that the total (*i.e.*, irrespective of traffic class) allocation. The user side implements the queue disciplines necessary for assuring the QoS of the various traffic classes. The class assignment is done on the user side, and although defaults for the most common profiles could be provided, the final assignment can be controlled via a configuration interface. The fact that the router does not do the classification, but only enforces the total allocation ensures the neutrality and fairness of the scheme.

The prototype implementation of the system runs on Linux hosts (both user and router functionalities). It implements the traffic policing component using the HTB [Dev03] packet scheduler system. Figure 5.8 illustrates the mapping of the allocations established by the demand-based marketplace into the parameters of a user-side two-class HTB. The root class enforces the physical rate of the bottleneck link. This is important when user systems are connected by a high-speed Ethernet, but share a much slower Internet connection.⁹ The HTB system is work-conserving, which means that if there are packets waiting in any of the queues and the output link is not being utilized, it will deliver the packets. Thanks to this feature, in the case where the traffic from one class exceeds its current allocation and the other is below, the system defaults to a standard best-effort service, limited only by the actual physical rate of the link (r in the illustration).

The market agent simply runs as a background process and adjusts the allocations (x_{ij}, w_{ij}) at the beginning of each time slot. It also communicates with the router's marketplace service ahead of the start of each *epoch* (*e.g.* a day) to negotiate the allocations for the next epoch. An important detail is that, if for some time slot the profile has $x_{ij} = 0$ and the demand from other users is very high, it may turn out

⁹In practice the system should implement two different 1st level classes, one for the local LAN traffic and the other for Internet traffic. We only present the simplified version.

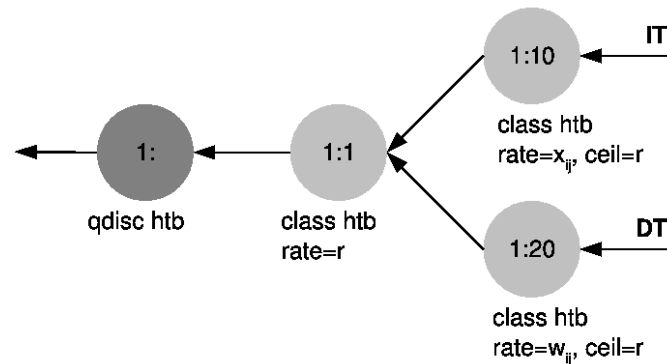


Figure 5-8: HTB-based implementation of the traffic policing component

that the allocation w_{ij} for this user is also zero, leaving no reservation for the user. To avoid this situation, our mechanism imposes a lower-threshold x_{min} to the profile, thus ensuring that all the users get at least some minimal allocation available at any time.

On the router side, the configuration of both the downstream and the upstream HTB “police agents” (responsible for the policing functionality) is a little different (from the router’s perspective these are egress agents). In this case there is a single police agent per user whose committed rate is the rate reserved for the timeslot, but the peak rate is the line-speed. This allows exploiting the work-conserving property of HTB: when the demand of some users is below their reservation, the available bandwidth is distributed among the other users in proportion to their reservations. The router does not do any packet classification, which minimizes overhead and makes the router operate in an application-agnostic manner, thus ensuring the neutral character of the service.

Algorithmic Complexity and Efficient Distributed Implementation: For a scheme like the one explained above to be practical, it needs to have a computationally efficient solution. In the case of fluid allocation, the solution using Lagrange multipliers presented in Section 5.3 constitutes a straightforward distributed

implementation, whereby at the Customer Premises Equipment (CPE) each agent computes its best response iteratively until the marketplace reaches a NE.

As for the trading of atomic tasks, the computation of best-response in the presence of constraints (such as preserving the ordering or the adjacency of session components) is a difficult problem. For its solution, a dynamic programming algorithm was devised. This algorithm is pseudo-polynomial (complexity depends on the product of the number of sessions per user and the number of time slots) and runs in a few seconds on current hardware for instances of practical sizes of hundreds of users and hundreds of time slots (108 and 288, respectively in the experimental evaluation of §5.7). The dynamic programming solution, when finding the best response for user i proceeds as follows:

1. Let k be number of sessions of user i , and T the number of time slots
2. Initialize the matrix A of dimension $k \times T$. Each element a_{jp} of A will represent the cumulative cost of sessions $1 \dots j \leq k$, when the j^{th} session is allocated in slot p . All the matrix elements are initialized to infinity.
3. The first row is computed by assuming session 1 is placed in slot p and computing the resulting cost.
4. As the user's sessions do not overlap, the cost of allocating each new session is given by the recurrence

$$a_{jp} = \begin{cases} \infty & \text{if session } j \text{ is unfeasible at slot } p \\ \min\{a_{j-1,1\dots p-1}\} + c(j,p) & \text{otherwise} \end{cases} \quad (5.24)$$

where $c(j,p)$ represents the cost of session j when allocated beginning at slot p . All the subsequent rows are computed using this recurrence and the minimum of the last row $\min\{a_{k,1\dots T}\}$ will give the optimal cost for the entire set of sessions of the user.

The feasibility condition in equation (5.24) refers to the constraints of the problem. Basically, all the components of the session fall within the allowed time slots ($1 \dots T$), and the cumulative cost is less or equal to the budget. The experimental evaluation did not include the capacity constraint, although it could be easily incorporated into the procedure.

5.7 Experimental Evaluation

This section presents a trace-driven evaluation of the T&C system that shows the benefits obtainable under real network workloads. In the first place, a brief description of the traces and the pre-processing done for the different experiments is done. Next, an evaluation of the demand-based fluid trading system is presented, followed by an experimental evaluation of the atomic trading system, and last an evaluation including the composition of both systems.

5.7.1 Traces and Trace Pre-Processing

Publicly available WAN traces [MAW09] were used to conduct the experimental evaluation of the system. These traces provide packet-level information on a high capacity WAN link. Table 5.1 summarizes the characteristics of this trace. For all the experiments, the traffic was aggregated in *5min* time slots, a sample of a *24hr* period was used, thus giving 288 time slots.

Period	2009-03-31 00:00 – 2009-03-31 23:59
Total packets	1,551,089,845
TCP packets	1,194,409,653
UDP packets	4,321,852
Total TCP bytes (payload)	924,540,189,060

Table 5.1: Characteristics of the WAN trace used in the evaluation.

The pre-processing steps used to extract the traffic associated with a customer access network were: First, identify subnets most likely associated with broadband

users, based on the upstream/downstream ratios, the activity per port number, and diurnal activity patterns. Next, assuming that each IP address is a single user/household, classify the traffic per user as either atomic or fluid. The general criteria used was that sessions due to interactive applications such as web-browsing, VoIP, media streaming are taken as interactive, and traffic due to P2P and other applications was assumed to be fluid. As in the atomic-trading phase considers the possibility that IT may also have some degree of delay-tolerance, the term Fluid-Traffic (FT) is used to refer specifically to fluid applications. The classification is done based on association of traffic activity with privileged port numbers. Research on traffic classification schemes has been very active recently (See for example [BTS06, KPF05, KBFc04, MZ05, MP05, SSW04, MPS09]) and potentially any of those schemes could be incorporated (*e.g.* as a library) in the implementation of the T&C marketplace. The last step is to identify the various IT sessions per user, with their corresponding demands per time slot. Session identification is done by setting a threshold on the length of periods of high activity. This threshold is called S_{max} and it is given as a number of time slots. For most of the experiments the values $S_{max} = 6$ and $S_{max} = 12$ were used. The first one corresponds to a half an hour period, and the second to a one hour period. If any sequence of time slots has length greater than S_{max} , then the minimum from this interval was subtracted under the assumption that it was due to FT. By repeating this process on any subinterval of length greater than S_{max} gives a set of disjoint IT sessions for the user. Figure 5-9 shows the aggregated inbound traffic for the selected subnetwork for both classes, IT and FT.

Budget assignment: To conduct the experiments, the budget was assigned so that the total budget accounts for the total traffic in the trace if transmitted at a constant rate. Then, the total budget was divided in n equal parts to assign the user budget.

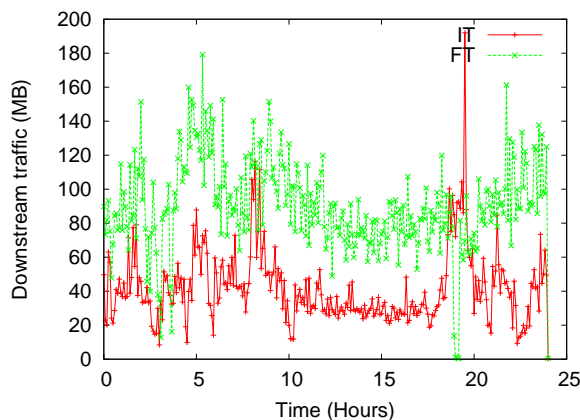


Figure 5-9: Downstream trace for a subnet of broadband users

5.7.2 Evaluation of the Demand-Based Fluid Trading System

The first set of experiments analyzes the effect of the fluid trading phase alone. In these experiments the IT components of the trace are the fixed allocations x_{ij} and the fluid component is dropped and allocated by the marketplace. The result of the experiment is illustrated in Figure 5-10. The IT traffic (whose allocation is preserved after the market clearing phase) is comparatively small. The curves for the total traffic (IT+FT) before the marketplace allocation shows high variability with peak throughput or about $274Mbps$. After the redistribution of FT traffic the total link reservation is essentially constant at about $138Mbps$. The 95% of the original traffic trace is $206Mbps$, therefore there is a significant reduction of peak utilization, which in cases where the cost is determined by the 95/5 rule means a significant reduction of cost as well. The fact that a significant fraction of the trace is P2P traffic (and thus handled as fluid allocation), made it possible to for the mechanism to achieve an almost perfect load-balance of the utilization.

Another important observation comes from comparing the total (over all time-slots) per user allocation of IT against its FT. Figure 5-11 shows this. Although the relationship looks linear, it is important to notice that it does not need to be so. For

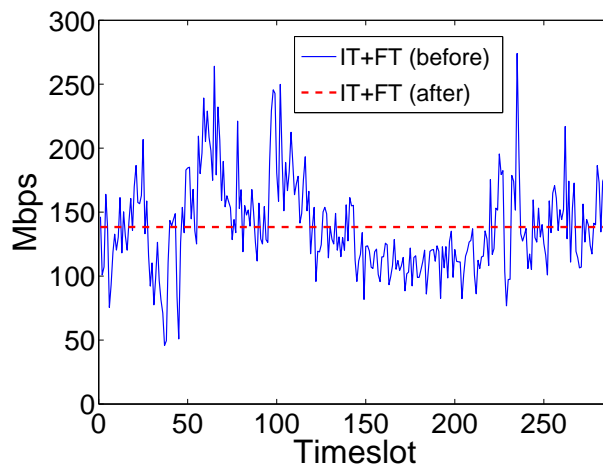


Figure 5·10: Traffic before and after marketplace reallocation

example two different users with the same amount of IT traffic, one of them at peak hours, while the other at off-peak hours will get very different amounts of FT. Notice though that the marketplace incentivizes users to declare their true IT profile. If a user declares less, well he/she may get less and his/her interactive applications will suffer. On the other hand, requesting more than necessary will never get the user more than what the optimal allocation does.

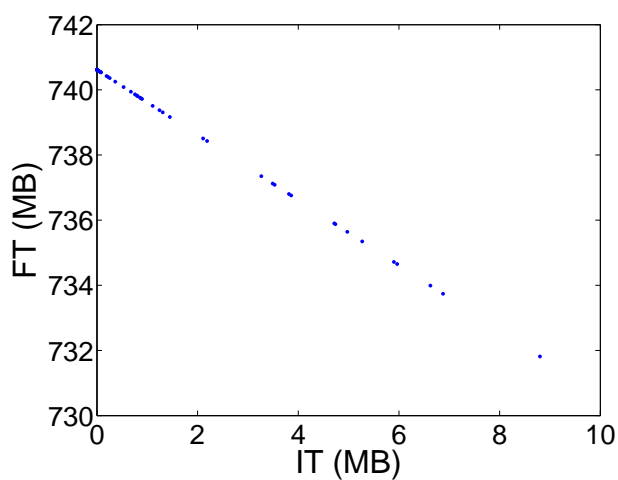


Figure 5·11: IT vs FT components per user

5.7.3 Evaluation of the Atomic Trading System

To test the atomic trading functionality, the IT components of the traces were taken as the fixed tasks input to the T&C marketplace. In addition, as network sessions typically span over multiple time slots, the constraint that the entire session had to be preserved (no breaking or re-ordering its components) was implemented. As a mechanism to indicate to the system the flexibility of IT sessions, a *slack* parameter was introduced. This parameter indicates the number of time slots a session may be moved back or forward in time. So for example, a slack of 0 implies no flexibility. A slack of 1 implies a willingness to shift sessions by one time slot (5 minutes in the traces) back or forth, if such a shift is advantageous. Notice that *moving a session* means a shift of the traffic attributed to that session for *all* time slots spanned by that session (*i.e.*, traffic in all time slots of a single session is shifted equally to preserve session atomicity). The simulations also enforce the condition that no shifting sessions could overlap. This is consistent with users not doing more activities on the same time slot.

The first experiment aims to evaluate how the 95th percentile of the link's 5-minute traffic volume (the 95% traffic envelop) changes as a result of letting users schedule their IT sessions. For brevity, it is assumed that all users adopt the same *slack* value for all their sessions. Figure 5-12 shows two examples of the outcome after the market reaches an equilibrium. On the left is the traffic per time slot, and on the right is the CDF of traffic per time slot. Top row is for session length threshold of $S_{max} = 6$, and the bottom row is for $S_{max} = 12$ time slots. Clearly, the session thresholding process has little effect on the trace, being the most noticeable effect the larger peak (from 130MB to 150MB). Table 5.2 shows the values of the 95% traffic envelop. These results underscore that selfishly scheduling IT sessions yields an equilibrium with *significant* reduction in the 95% traffic envelop – up to

31% reduction when slack is 1 hours. Even for a small slack of 15 minutes, the savings amount to 16%.

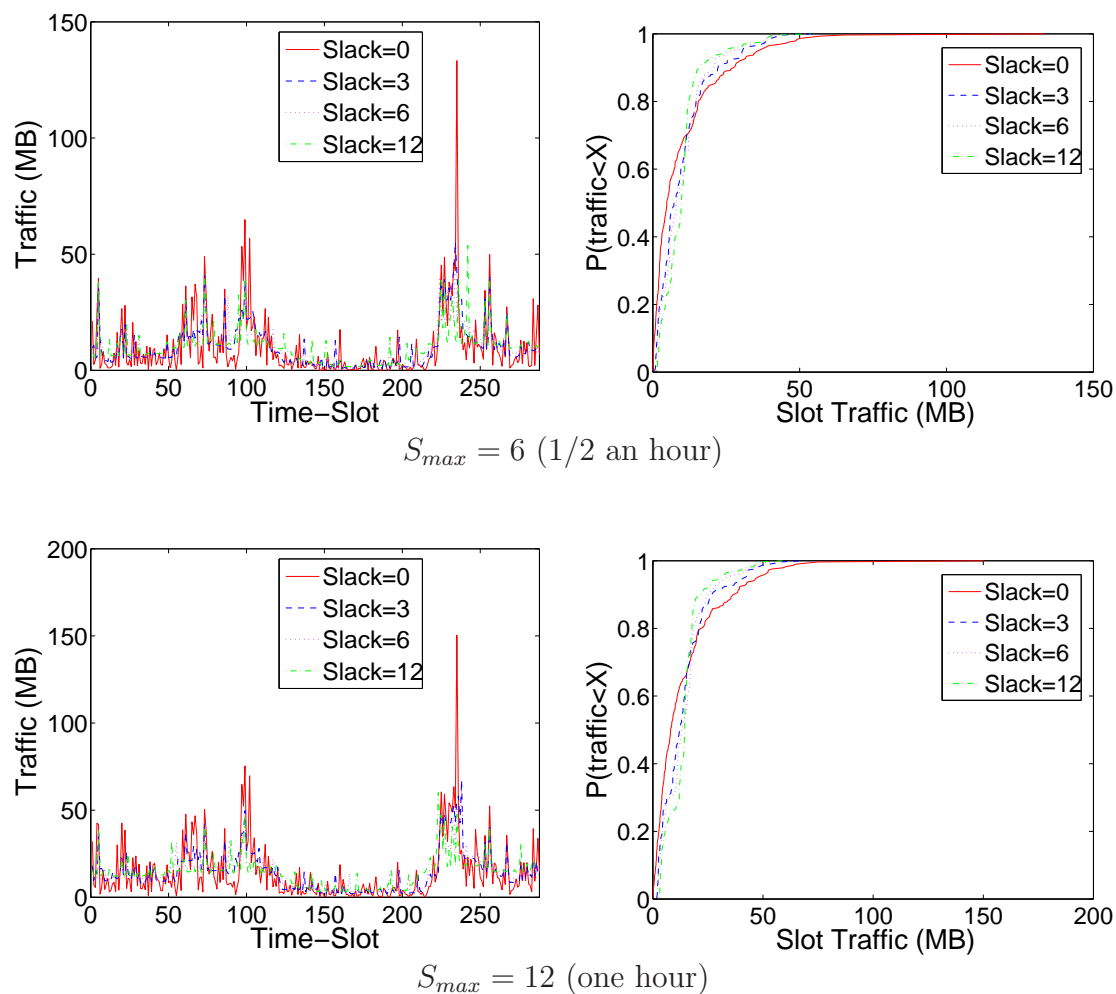


Figure 5-12: Utilization over time for IT sessions with various slack values.

5.7.4 Evaluation of Combined Atomic and Fluid Trading

The last set of experiments considers both phases of T&C. In particular, after completing the trading phase – thus scheduling all IT sessions in the trace – users allocate as much fluid traffic as possible in accordance with their remaining budgets. Thus, an important consideration in setting-up these experiments is the budget

<i>Slack</i>	$S_{max} = 6$		$S_{max} = 12$	
	95%(<i>MB</i>)	Reduction%	95%(<i>MB</i>)	Reduction%
0	36.3	0.0	47.7	0.0
3	30.6	15.6	42.1	11.7
6	27.4	24.4	33.6	29.6
12	24.9	31.4	30.9	35.2

Table 5.2: 95% utilization resulting from bandwidth trading.

assignment. In particular, the following policy was used: Let V denote the nominal traffic per time slot that results in a total volume equal to the total traffic originally in the trace. Let R (for resistance) be a control parameter which allows the provider to adjust the resulting traffic on the shared link. By setting $C = V/R$ (C is the constant used for normalizing utilization, see §5.2), and the budget per customer to $B_i = CT/n$, the expected utilization (without IT) is precisely C .

Figure 5-13 shows the outcome of the two phases of T&C for a value of $R = 1.0$ and various slack values. The y-axis is normalized with respect to V (the nominal volume under perfectly balanced conditions, with no IT components). Due to the presence of IT components, this quantity is always (slightly) larger than 1.0. The session identification process also capture a much larger peak in the case of $S_{max} = 12$. Table 5.3 shows the 95% and 50% (median) of the time slot utilizations, as well as the ratio between them. These results suggest that with T&C in place, the ratio is nearly 1.0, resulting in a perfect flattening of traffic over time slots, thus eliminating cost problems derived from spikes when using the 95/5 rule.

		95%	Median	Ratio
Original		197.15	124.56	1.583
T&C	$S_{max} = 6$	136.52	135.93	1.004
T&C	$S_{max} = 12$	138.05	137.33	1.005

Table 5.3: Traffic volume statistics (in MB) with and without T&C

The parameter R allows the provider to specify a target total traffic volume on the managed link. Figure 5-14a shows the total allocation per time slot as a function of R , when $slack=0$ (which is the worst-case in the sense that under this scenario

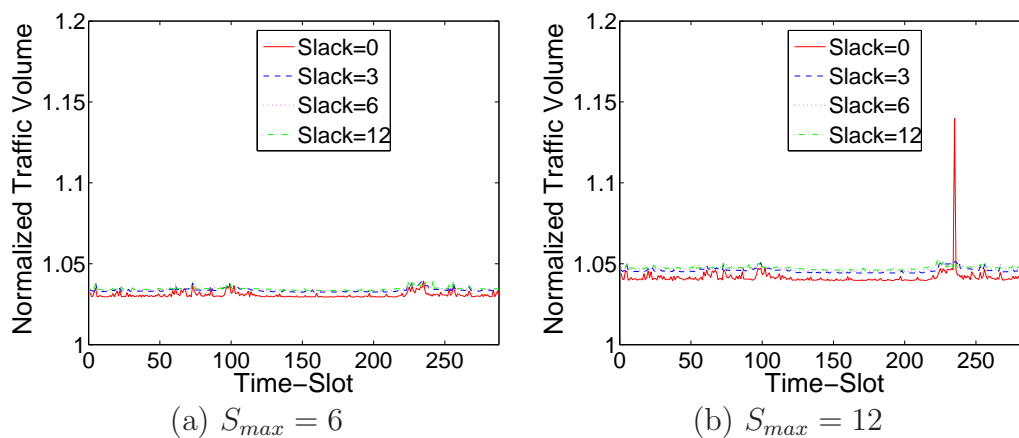
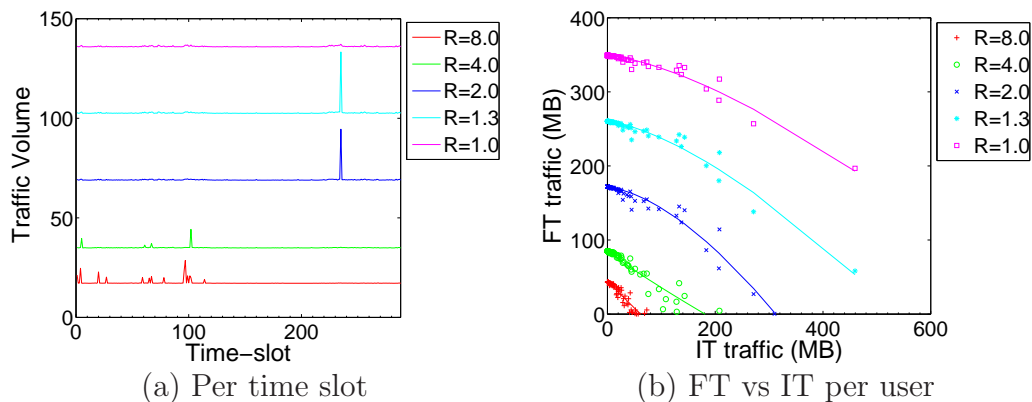


Figure 5-13: Total Traffic (IT+FT) for various slack values.

there is no IT flexibility). As expected, R effectively controls the aggregate traffic volume resulting from T&C. This volume is almost flat due to the “fluid” nature of FT bandwidth allocation. The exception is due to spikes underscoring the presence of large IT sessions that could not be smoothed out under the chosen slack value. Naturally, these spikes dissipate when larger slack values are used (see Figure 5-13).

The impact of the allocation of FT relative to IT is evaluated in Figure 5-14b, which compares the per-user bandwidth allocations for different values of the resistance, R . As before, the general trend is that the more IT bandwidth requested by a user during the trading phase, the less FT allocation the user is able to secure during the capping phase. Increasing the values of R results in a corresponding reduction in the aggregate allocation of FT bandwidth, with large IT bandwidth consumers impacted the most.

To evaluate T&C on a per-user basis, a comparison on how IT and FT allocations vary across users was done. Figure 5-15 (left) shows a clear negative correlation between the allotment of FT and IT bandwidth. The relationship is not monotonic or deterministic because it depends on the outcomes from the atomic phase, which affect the left-over budget for each player. It is always the case though that the larger the slack, the larger the FT allocation for any given user (points along the same

Figure 5-14: Traffic allocations for variable R .

vertical line in the plot). An agent with fixed IT demand increases its allocation of FT bandwidth when it adds more flexibility to its IT sessions. The results in Table 5.4 expose this tradeoff for selected levels of IT demand and resistances. For example, when $R = 4$, a user with a nominal 100MB of IT bandwidth is able to capture 32% more FT traffic by accepting a minimal slack of 3 for its IT sessions. A rather surprising (and also desirable) finding – evident from Figure 5-15 and Table 5.4 – is that the user begets *most* of the benefit by introducing a minimal amount of slack. Increasing the slack beyond that, results in only marginal increases in FT allocation. In the above example, by doubling its slack from 3 to 6, the user is able to capture only 3% more FT traffic. Figure 5-15 (right) shows the same results on a semi-log scale to expose the outcome for users with negligible demand for IT bandwidth. In this case, the capping phase assigns to all such users almost equal share of the capacity (as expected). It is only the heavy IT bandwidth hogs who are unable to claim much FT bandwidth, which is precisely the premise of T&C.

	R=4.0	R=2.0	R=1.0
Slack	100MB	200MB	400MB
3	1.3190	1.2836	1.1931
6	1.3497	1.3338	1.2329
12	1.4079	1.3769	1.2520

Table 5.4: FT gain for various values of R and IT demand.

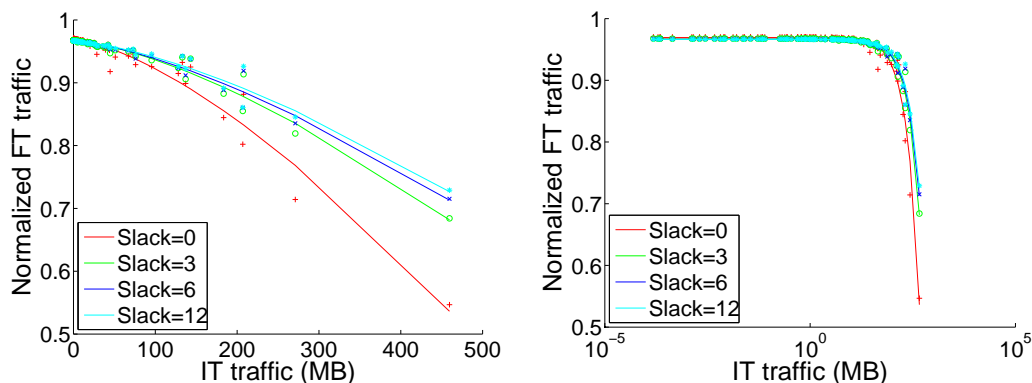


Figure 5-15: IT and FT allocations per user for different slack values.

5.8 Related Work

The problem of allocating the traffic of multiple users to various traffic classes when using a QoS mechanism is interesting because selfish users are willing to present all their traffic as high priority traffic to the system. This problem has been considered from a game-theoretic perspective by various works, for example [Mar04, PSC98, CP99]. Marback [Mar04] analyzes a priority queueing scheme where packets get charged based on their priority, and selfish users compete for bandwidth. Among other things, he shows that such a scheme leads to a Wardrop equilibrium and that allocation does not depend on the prices of each traffic class. A fundamental distinction with respect to T&C is that T&C enables different valuations for different classes of traffic, and uses these valuations to leverage the trading system. Park *et al* [PSC98] consider a QoS class assignment game where users share a single Generalized Processor Sharing (GPS) queue and they can assign the class for the traffic. Users do so, to meet the QoS requirements of their application at the minimum possible costs (as higher priority also means higher cost). In this work, they consider both, the case where traffic may be arbitrarily split between the many service classes and the *unsplittable* case where all the traffic is assigned to the same class. In the splittable case, NE need not exist, but it is proven that in the unsplittable case NE always

exists. Chen *et al* [CP99] consider the assignment of service classes to each user's traffic at each one of the routers in a path. In this analysis, each user provides a QoS vector and a utility function, and the user actions are the choices of service classes at each router, such that his traffic will meet the QoS goals with minimum cost. This model is limited to the *unsplittable case*, meaning that all the traffic from a user is assigned the same service class. The incentive for the users is implicit in the price-by-class scheme, where users requesting higher priority classes pay more. In addition, payment has to be made to all intermediary nodes on a route. Chen *et al* [CP98] also provide an efficient distributed implementation and evaluation of their multi-switch QoS assignment game, where agents running at the routers and end-points compute the game outcome on behalf of the users. The performance evaluation shows a significant improvement on the per-application QoS metrics with respect to a static reservation mechanism. A common characteristic of these set of works is that they focus in situations where a set of concurrent flows share the resource, as opposed to T&C where the schedule of tasks of different classes creates the tradeoff for the users to reallocate their tasks in such a manner that improves the performance for different traffic classes (response times, aggregated throughput).

Approaches for *congestion-pricing* with explicit payments have been considered in a number of studies. Henderson *et al* [HCB01] present a review of the benefits and limitations of these proposals. Examples include *Smart Markets* [MMV95b, MMV95a] and *Split-Edge Pricing* [Bri99]. Of particular interest is the scheme proposed by Ganesh *et al* [GLS01], which assigns costs to packets depending on congestion. Under a family of non-linear cost functions that depend on the utilization of the congested link and the flow's demand, they showed convergence to steady-state equilibrium. While our mechanism and system model are entirely different, our cost function has similar characteristics. A fundamental difference in our scheme is that

the tradeoff for the users comes from the schedule of their tasks, and it is not an instantaneous congestion-control mechanism. In fact, the T&C marketplace is intended to serve as a high level meta-scheduler between the applications sharing the resource.

Laoutaris and Rodriguez [LR08], note that the problems associated with large demands for DT traffic could be alleviated if DT traffic were scheduled over periods of low demand. However, systems currently deployed lack incentive mechanisms to do so, and the low-demand periods are typically not synchronized when a network path traverses several time zones. As a solution to the first problem, they suggest giving users “higher-than-purchased” access rate during off-peak hours as a reward for time-shifting their DT traffic. As a solution to the second problem, they propose the introduction of a store-and-forward service to handle the network transfer of bulk DT data during off-peak hours. This second model is further developed in [LSRS09], where several algorithms are proposed for scheduling the delivery of DTB traffic across a wide area network. These mechanisms address the problem when the tasks are under the control of a single authority that can optimize and schedule the execution of the tasks accordingly. The T&C marketplace addresses the problem when multiple non-cooperative agents share the resource.

5.9 Summary

This chapter presents the T&C marketplace as a coordination mechanism for allowing the load-balancing of a common resource shared by multiple selfish and rational parties. The mechanism incentivizes users to schedule their delay-tolerant tasks at times of low demand, and the incentive is delivered in terms of the amount of resource allocation provided, enabling the T&C marketplace to operate in environments where currency exchange is not possible.

The analysis of T&C shows that in the case of a fluid-task market, there is a single global optimum which is also an equilibrium for the users. In the case of atomic-task market, there may be multiple Nash Equilibria, but a better-response bidding process is proven to converge to the equilibrium. Although the Price of Anarchy associated with this process is n , in practice when the number of users (and tasks) involved is large, it is typically very small. The combination of both atomic and fluid markets is possible and its solution is again an equilibrium that achieves a load-balanced system, limited only but the tasks with no flexibility.

The experimental evaluation of the system shows that in a real-life scenario T&C can achieve a significant reduction of the peak-to-valley ratio of the workload over time. This is particularly interesting in cases where service rates are dependent on the peak utilization, as it is the case in bandwidth and energy markets, for example.

Chapter 6

Conclusion

Along with the deployment of wide-area network infrastructures, many new services and applications have come into existence. These services rely on sharing resources for efficiency, but the sharing of resources raises many challenging problems as well. One of the most fundamental problems is the network embedding problem: finding and allocating resources for the applications on a given infrastructure. In this thesis we presented contributions on three facets of the problem: 1) when users need to find feasible sets of resources that meet the requirements of their applications, 2) when users reserve fractions of shared resources, and share the costs in proportion to their reserved fractions, and 3) when the users' applications share resources over time, but the lack of coordination is detrimental for both the users and the provider.

6.1 Summary of Contributions

Our first contribution, NETEMBED addresses the first facet. NETEMBED includes a language for describing the applications' requirements and the infrastructure capabilities, and mapping algorithms for finding feasible allocations of resources to applications. NETEMBED's algorithms rely on constraint-satisfaction techniques, which are proven to be free of false positives and free of false negatives. To address the scalability problem, NETEMBED introduces three search algorithms, Exhaustive Search with Constraint Filtering (ECF), Random Walk with Backtracking (RWB), and Lazy Neighborhood Search (LNS), exploiting two techniques to speed-up the

search: 1) *early pruning* – removing unfeasible regions of the search space as soon as possible, 2) *candidate reordering* – by organizing the candidate sets according to their cardinality, the number of steps in the search process is minimized. The experimental evaluation of these techniques shows that they scale to problem instances in the range of thousands of nodes and links, with run times in the order of tens of seconds. The experimental evaluation also shows that NETEMBED’s algorithms are complementary in the sense that ECF and RWB are better suited to handle very constrained problems, while LNS performs better on problems with little constraints or in which the constraints are loose (in the sense that they do not help prune many candidates).

Our second contribution, the *Colocation Games*, address the problem of allocating fractional capacities of the resources to different users and sharing the costs in proportion to the fraction allocated by each user. Our analysis shows that for the most general version of Colocation Game a Nash Equilibrium (NE) does not necessarily exist. In contrast, the more restricted Process Colocation Game and Uniform Parallel Process Colocation Game are shown to converge to a NE under better-response dynamics. For these cases, bounds on the Price of Anarchy with respect to the utilization of resources were proven. These results also extend to cases where resources are described by multidimensional resource capacities, the Multidimensional Process Colocation Game. The existence and convergence to a NE, the fact that the PoA is bounded, and the experimental analysis that shows that in practice under large numbers of users the outcome achieves high-utilization; let us conclude that these variants of the *Colocation Games* serve as practical allocation mechanisms in scenarios where applications need performance guarantees and overall high-utilization yields high system efficiency.

Our third contribution, the Trade & Cap (T&C) marketplace addresses the prob-

lem of sharing a resource over time, while minimizing the maximum utilization of the resource. T&C creates a marketplace where allocation shares may be traded by users, each one maximizing its own benefit. This mechanism is intended to be used in scenarios where the resource is under a central control, but the scheduling of tasks is at the discretion of the users. In many real systems pricing mechanisms are fixed externally and cannot be changed. Instead, in T&C users trade allocation shares whose value is driven by forces of supply and demand. In so doing, users get an incentive in the form of a larger allocation if they schedule their delay-tolerant tasks during periods of low demand. Fundamental properties of the marketplace such as the existence of equilibria, the convergence, and the efficiency (in terms of balanced load) were established analytically and evaluated experimentally, showing that for both the users and the provider there are significant gains. The users benefit by obtaining a larger aggregate allocation for delay-tolerant applications, and by improving the QoS of interactive applications. The provider benefits as better balance of system utilization translates into lower operating and provisioning costs.

6.2 Future Directions

Large distributed systems are essential elements in many applications running on today's Internet. Automating the resource management of such systems is fundamental to cope with their scale and to make efficient use of the available resources. However, accomplishing this objective is non-trivial because of the contrasting interests of the various parties involved. Nevertheless, efficiency is a common goal for both sides: Achieving better utilization of the resources improves the profitability for providers and lowers the cost for the consumers. A fundamental challenge in the area is the design of mechanisms that allow the parties to realize this synergy.

Standard mechanisms from micro-economics are not necessarily applicable for

the management of computing resources. For example, common assumptions such as explicit knowledge of private valuations, that valuations can be expressed using a common currency, and that the parties are willing to disclose their valuations, do not always hold. T&C drops the first two. The goal of the mechanism can be quite different as well. Micro-economic mechanisms commonly have the goal of maximizing the *social value*, understood as the summation of the valuations of the winners (users who get the resources). In the case of computing resources, when those valuations are not explicitly known, this goal would be difficult to achieve. Instead, some form of fairness, understood as an equitable (or proportional) distribution of the resources among the users of the system is frequently preferred. There is then a large spectrum of resource management problems where users of the system are essentially uncooperative, but a mechanism that gives the right incentives may be used to ensure overall efficiency and fairness.

In designing such mechanisms, many additional problems come into play as well. For example how to enforce that the outcome of the mechanism is carried out (all the parties comply with their commitments); if it is possible to implement the mechanism relying only on local information, so that the mechanism can be implemented when agents interact in a decentralized manner; or if the mechanism provides the means to handle faulty or byzantine agents. The study of these problems is collectively known as *Distributed Algorithmic Mechanism Design*, and although there are some promising results, there are also many questions still open [FS02].

Bibliography

- [AAE05] Baruch Awerbuch, Yossi Azar, and Amir Epstein. The price of routing unsplittable flow. In *STOC'05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 2005. ACM. doi:10.1145/1060590.1060599.
- [ABC⁺09] Alvin AuYoung, Phil Buonadonna, Brent N. Chun, Chaki Ng, David C. Parkes, Jeff Shneidman, Alex C. Snoeren, and Amin Vahdat. *Market Oriented Grid and Utility Computing*, chapter Two Auction-Based Resource Allocation Environments: Design and Experience. Wiley, 2009.
- [ABKM01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, 2001. doi:10.1145/502034.502048.
- [ACSV04] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the Ondemand IT InfraStructure*, October 2004.
- [AD54] K. J. Arrow and G. Debreu. The existence of an equilibrium for a competitive economy. *Econometrica*, XXII(3):265–290, 1954.
- [ADK⁺04] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Foundations of Computer Science (FOCS04)*, 2004.
- [ALR03] Chris Alfeld, Jay Lepreau, and Robert Ricci. A solver for the network testbed mapping problem. *SIGCOMM Computer Communications Review*, 33(2):65–81, 2003. doi:10.1145/956981.956988.
- [AM04] Lawrence M. Ausubel and Paul Milgrom. *Combinatorial Auctions*, chapter 1: The Lovely but Lonely Vickrey Auction. MIT Press, 2004.
- [Ama09] Amazon.com, Inc. Amazon EC2 Spot Instances. Web, December 2009. Available from: <http://aws.amazon.com/ec2/spot-instances/>.

- [AOPV08] Jeannie Albrecht, David Oppenheimer, David Patterson, and Amin Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. *ACM Transactions on Internet Technology*, 8(4):1–44, May 2008. doi:10.1145/1391949.1391952.
- [BCMR02] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *Proceedings of ACM SIGCOMM*, pages 47–60, Pittsburgh, PA, August 2002. doi:10.1109/TNET.2004.836103.
- [BEH⁺01] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M.S. Marshall. GraphML Progress Report: Structural Layer Proposal. In Springer-Verlag, editor, *Proceedings of the 9th International Symposium on Graph Drawing (GD '01), LNCS 2265*, pages 501–512, 2001. Available from: <http://graphml.graphdrawing.org/specification.html>.
- [BH09] Luiz André Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009. doi:10.2200/S00193ED1V01Y200905CAC006.
- [BLBS03] Suman Banerjee, Seungjoon Lee, Bobby Bhattacharjee, and Aravind Srinivasan. Resilient multicast using overlays. In *Proceedings of ACM SIGMETRICS*, June 2003. doi:10.1145/885651.781041.
- [Bri99] Bob Briscoe. The direction of value flow in connectionless networks. In *Networked Group Communication*, pages 244–269, 1999. doi:10.1007/b72228.
- [Bri07] Bob Briscoe. Flow rate fairness: Dismantling a religion. *SIGCOMM Computer Communications Review*, 37(2):63–74, April 2007. doi:10.1145/1232919.1232926.
- [BTS06] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, New York, NY, USA, 2006. ACM. doi:10.1145/1368436.1368445.
- [CBMp03] Jeffrey Considine, John W. Byers, and Ketan Mayer-patel. A constraint satisfaction approach to testbed embedding services. In *Proceedings of HotNets-II*, November 2003. doi:10.1145/972374.972398.
- [CDK⁺03] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *SOSP'03*, Lake Bolton, NY, October 2003.

- [CFH⁺05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
- [CNA⁺04] Brent N. Chun, Chaki Ng, Jeannie Albrecht, David C. Parkes, and Amin Vahdat. Computational resource exchanges for distributed resource allocation, 2004.
- [CP98] Shaogang Chen and Kihong Park. A distributed protocol for multi-class QoS provision in noncooperative many-switch systems. In *Proceedings of the Sixth International Conference on Network Protocols*, pages 98–107, Oct 1998. doi:10.1109/ICNP.1998.723730.
- [CP99] Shaogang Chen and Kihong Park. An architecture for noncooperative QoS provision in many-switch systems. In *Proceedings of IEEE INFOCOM*, volume 2, pages 864–872, Mar 1999. doi:10.1109/INFCOM.1999.751475.
- [CR06] Ho-Lin Chen and Tim Roughgarden. Network design with weighted players. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 29–38, New York, NY, USA, 2006. ACM. doi:10.1145/1148109.1148114.
- [Cro07] Jon Crowcroft. Net neutrality: the technical side of the debate: a white paper. *SIGCOMM Computer Communications Review*, 37(1):49–56, Jan 2007. doi:10.1145/1198255.1198263.
- [CRSZ02] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–71, October 2002. doi:10.1109/JSAC.2002.803066.
- [CV02] Artur Czumaj and Berthold Vöcking. Tight bounds for worst-case equilibria. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 413–420, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.
- [Dev03] Martin Devera. HTB Home. Web page, July 2003. Available from: <http://luxik.cdi.cz/~devik/qos/htb/>.

- [DT09] Christoph Dürr and Nguyen Kim Thang. Non-clairvoyant scheduling games. In *Proceedings of the 2nd International Symposium on Algorithmic Game Theory (SAGT)*, 2009.
- [DZH03] Zhenhai Duan, Zhi-Li Zhang, and Yiwei Thomas Hou. Service overlay networks: SLAs, QoS, and bandwidth provisioning. *IEEE/ACM Transactions on Networking*, 11(6):870–883, December 2003. doi:10.1109/TNET.2003.820436.
- [FCC⁺03] Yun Fu, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. SHARP: An architecture for secure resource peering. In *Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP'03)*, volume 37, pages 133–148, Dec 2003. doi:10.1145/1165389.945459.
- [FKK⁺02] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The Structure and Complexity of Nash Equilibria for a Selfish Routing Game. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 123–134, Malaga, Spain, 2002.
- [FKNT99] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.
- [Fol07] Dagfinn Follesdal. The status of rationality assumptions in interpretation and in the explanation of action. *Dialectica*, 36(4):301–316, Jun 2007. doi:10.1111/j.1746-8361.1982.tb01545.x.
- [FPS01] Joan Feigenbaum, Christos H. Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001. doi:10.1006/jcss.2001.1754.
- [FS02] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications (DIALM)*, pages 1–13, New York, NY, USA, 2002. ACM. doi:10.1145/570810.570812.
- [GEN10] The GENI initiative. Online, 2010. Available from: <http://www.geni.net/>.
- [GHMP09] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: Research problems in data center networks. *CCR Online*, Jan 2009.

- [GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, CA, 1979.
- [GLS01] A. Ganesh, K. Laevens, and R. Steinberg. Congestion pricing and user adaptation. In *Proceedings of IEEE INFOCOM*, pages 959–965, 2001. doi:10.1109/INFCOM.2001.916288.
- [GNY04] X. Gu, K. Nahrstedt, and B. Yu. SpiderNet: An Integrated Peer-to-Peer Service Composition Framework. In *Proceedings. 13th IEEE International Symposium on High performance Distributed Computing*, pages 110–119, June 2004. doi:10.1109/HPDC.2004.1323507.
- [Hay08] Brian Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008. doi:10.1145/1364782.1364786.
- [HCB01] T. Henderson, J. Crowcroft, and S. Bhatti. Congestion pricing. Paying your way in communication networks. *IEEE Internet Computing*, 5(5):85–89, Sep/Oct 2001. doi:10.1109/4236.957899.
- [HHK⁺01] Joseph Hall, Jason Hartline, Anna R. Karlin, Jared Saia, and John Wilkes. On algorithms for efficient data migration. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 620–629, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
- [Hud] Scott E. Hudson. CUP: LALR Parser Generator for Java. Online. Available from: <http://www2.cs.tum.edu/projects/cup/>.
- [JFI] JFlex – The Fast Scanner Generator for Java. Online. Available from: <http://www.jflex.de/>.
- [JM07] K. Jain and M. Mahdian. *Algorithmic Game Theory*, chapter Cost Sharing. In Nisan et al. [NRTV07], 1st edition, 2007.
- [KB04] Gu-In Kwon and John W. Byers. ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004. doi:10.1109/INFCOM.2004.1354511.
- [KBFc04] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of P2P traffic. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, New York, NY, USA, 2004. ACM. doi:10.1145/1028788.1028804.

- [KP99] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563, pages 404–413, 1999. doi:10.1007/3-540-49116-3.
- [KPF05] Thomas Karagiannis, Dina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *Proceedings of SIGCOMM*, pages 229–240, New York, NY, USA, 2005. ACM. doi:10.1145/1080091.1080119.
- [KRAV03] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 282–297, New York, NY, USA, 2003. ACM. doi:10.1145/945445.945473.
- [Leu04] Joseph Y-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.
- [LR08] Nikolaos Laoutaris and Pablo Rodriguez. Good Things Come to Those Who (Can) Wait or how to handle Delay Tolerant traffic and make peace on the Internet. In *Proceedings of HotNets*, 2008.
- [LRA⁺05] Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Systems*, 1(3):169–182, Aug 2005.
- [LSBB07] Nikolaos Laoutaris, Georgios Smaragdakis, Azer Bestavros, and John W. Byers. Implications of selfish neighbor selection in overlay networks. In *Proceedings of IEEE INFOCOM 2007*, Anchorage, AK, May 2007.
- [LSRS09] Nikolaos Laoutaris, Georgios Smaragdakis, Pablo Rodriguez, and Ravi Sundaram. Delay tolerant bulk data transfers on the internet. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 229–238, New York, NY, USA, 2009. ACM. doi:10.1145/1555349.1555376.
- [LZZ⁺04] Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the Internet. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS '04)*, San Diego, CA, February 2004.
- [Mar04] Peter Marbach. Analysis of a static pricing scheme for priority services. *IEEE/ACM Transactions on Networking*, 12(2):312–325, Apr 2004. doi:10.1109/TNET.2004.826275.

- [MAW09] MAWI Working Group. Traffic archive. Online, 2009. Available from: <http://tracer.csl.sony.co.jp/mawi/>.
- [MFPA09] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102, New York, NY, USA, 2009. ACM. doi:10.1145/1644893.1644904.
- [MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user’s perspective. Technical Report 2001-003, Boston University, 2001.
- [MMV95a] J.K. MacKie-Mason and H.R. Varian. Pricing congestible network resources. *IEEE Journal on Selected Areas in Communications*, 13(7):1141–1149, Sep 1995. doi:10.1109/49.414634.
- [MMV95b] J.K. MacKie-Mason and H.R. Varian. *Public Access to the Internet*, chapter Pricing the Internet, pages 269–314. MIT Press, 1995.
- [MP05] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Network Measurement (PAM'05)*, pages 41–54, 2005. doi:10.1007/b135479.
- [MPS09] Marco Mellia, Antonio Pescapè, and Luca Salgarelli. Traffic classification and its applications to modern networks. *Computer Networks*, 53(6):759–760, April 2009. doi:10.1016/j.comnet.2008.12.007.
- [MZ05] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of ACM SIGMETRICS*, pages 50–60, New York, NY, USA, 2005. ACM. doi:10.1145/1064212.1064220.
- [Nas51] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):268–295, Sep 1951.
- [NRTV07] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 1st edition, 2007.
- [NS209] The network simulator - NS2. Online, 2009. Available from: <http://www.isi.edu/nsnam/ns/>.
- [OAPV05] David Oppenheimer, Jeannie Albrecht, David Patterson, and Amin Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *14th IEEE Symposium on High Performance Distributed Computing (HPDC-14)*, July 2005. doi:10.1109/HPDC.2005.1520946.

- [Odl01] Andrew Odlyzko. Internet pricing and the history of communications. *Computer Networks*, 36:493–517, 2001. doi:10.2139/ssrn.235283.
- [OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.
- [PACR03] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Computer Communications Review*, 33(1):59–64, 2003. doi:10.1145/774763.774772.
- [Pap01] Christos Papadimitriou. Algorithms, games, and the internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753, New York, NY, USA, 2001. ACM. doi:10.1145/380752.380883.
- [PP06] KyoungSoo Park and Vivek S. Pai. CoMon: a mostly-scalable monitoring system for planetlab. *SIGOPS Operating Systems Review*, 40(1):65–74, 2006. doi:10.1145/1113361.1113374.
- [PSC98] Kihong Park, Meera Sitharam, and Shaogang Chen. Quality of service provision in noncooperative networks: heterogenous preferences, multi-dimensional QoS vectors, and burstiness. In *ICE '98: Proceedings of the first international conference on Information and computation economies*, pages 111–127, New York, NY, USA, 1998. ACM. doi:10.1145/288994.289022.
- [Ros73] Robert W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, December 1973. doi:10.1007/BF01737559.
- [RT02] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002. doi:10.1145/506147.506153.
- [RWE⁺01] Sean Rhea, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon and, and John Kubiawicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, September/October 2001. doi:10.1109/4236.957894.
- [San96] Tuomas Sandholm. Limitations of the vickrey auction in computational multiagent systems. In *In Proceedings of the Second International Conference on Multiagent Systems (ICMAS'96)*, pages 299–306. AAAI Press, 1996.

- [SDR04] Emil Sit, Frank Dabek, and James Robertson. UsenetDHT: A low overhead usenet server. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS '04)*, San Diego, CA, February 2004.
- [Sha53] Lloyd S. Shapley. A value for n-person games. *Contributions to the theory of games*, 2:31–40, 1953.
- [She95] Scott Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communications*, 13(7):1176–1188, September 1995. doi:10.1109/49.414637.
- [SSW04] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of P2P traffic using application signatures. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 512–521, New York, NY, USA, 2004. ACM. doi:10.1145/988672.988742.
- [STZ04] Subhash Suri, Csaba Tóth, and Yunhong Zhou. Uncoordinated load balancing and congestion games in P2P systems. In *Proceedings of IPTPS*, 2004.
- [Sun96] Rangarajan K. Sundaram. *A First Course in Optimization Theory*. Cambridge University Press, 1996.
- [TV07] É. Tardos and V. Vazirani. *Algorithmic Game Theory*, chapter Basic Solution Concepts and Computational Issues. In Nisan et al. [NRTV07], 1st edition, 2007.
- [web] CoMon: A monitoring infrastructure for PlanetLab. Online. Available from: <http://comon.cs.princeton.edu/>.
- [WLG02] B. White, J. Lepreau, and S. Guruprasad. Lowering the barrier to wireless and mobile experimentation. In *In Proceedings of Hotnets-I*, Princeton, NJ, October 2002.
- [WLS⁺02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of OSDI*, pages 255–270, Boston, MA, December 2002. USENIX Association. doi:10.1145/844128.844152.
- [Ye08] Yinyu Ye. A path to the Arrow-Debreu competitive market equilibrium. *Mathematical Programming*, 111(1):315–348, 2008. doi:10.1007/s10107-006-0065-5.

- [YL05] Tao Yu and Kwei-Jay Lin. A Broker-Based Framework for QoS-Aware Web Service Composition. In *2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, volume 00, pages 22–29, Los Alamitos, CA, USA, 2005. IEEE Computer Society. doi:10.1109/EEE.2005.1.
- [Yos06] Chad Yoshikawa. All-sites-pings for planetlab. Web, 2006. Available from: <http://ping.ececs.uc.edu/ping/>.
- [YYRC08] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Computer Communications Review*, 38(2):17–29, 2008. doi:10.1145/1355734.1355737.
- [YZL07] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Transactions on the Web*, 1(1):6, 2007. doi:10.1145/1232722.1232728.
- [ZA06] Yong Zhu and Mostafa Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proceedings of IEEE INFOCOM*, pages 1–12, Barcelona, Spain, April 2006. doi:10.1109/INFOCOM.2006.322.
- [ZCB96] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM*, 1996.

Curriculum Vitae

Jorge Mario Londoño Peláez

Personal

- Born on March 29, 1970.
- Colombia Citizen.
- Contact at jorge.londono@gmail.com

Education

- B.S. Electronics Engineering, Universidad Pontificia Bolivariana, 1992.
- M.A. Computer Science, Boston Univeristy, 1999.
- Ph.D. Candidate Computer Science, Boston University, 2010.

Employment

- Research Assistant, Boston University 2008–2010.
- Professor, Universidad Pontificia Bolivariana 1993–present.

Publications

- Jorge Londoño, Azer Bestavros, and Shang-Hua Teng. Colocation Games and Their Application to Distributed Resource Management. In *HotCloud'09*, San Diego, CA. 2009.
- Jorge Londoño and Azer Bestavros. A Two-Tiered On-Line Server-Side Bandwidth Reservation Framework for the Real-Time Delivery of Multiple Video Streams. In *Multimedia Computing and Networking 2009 (MMCN 2009)*, San Jose, California. January, 2009.

- Jorge Londoño and Azer Bestavros. NETEMBED: A Network Resource Mapping Service for Distributed Applications. In *Proceedings of the IEEE/ACM IPDPS High-Performance Grid Computing Workshop (HPGC08)*, Miami, Florida. April 2008.
- Jorge Londoño and Azer Bestavros. NETEMBED: A Service for Embedding Distributed Applications (Extended Abstract and Poster). In *Proceedings of ACM/IFIP/Usenix Middleware Conference (Middleware07)*, Newport Beach, California. November 2007.