A Middleware Framework for Self-Adaptive Large Scale Distributed Services

Dissertation submitted for the degree of

Doctor of Philosophy

by

Pablo Chacín Martínez

Advisor

Dr. Leandro Navarro Moldes

Universitat Politécnica de Catalunya

Departament d'Arquitectura dels Computadors

June, 2011

To my son Adrián, for contacting me with the joy of life and helping me to put things into perspective.

To Christiane, my mentor and friend, for being my inspiration as a computer scientist and a teacher.

Acknowledgments

Pursuing a Ph.D is an extraordinary and challenging endeavor that would not be possible to complete without the involvement of many people.

First and foremost, I am profoundly indebted to Dr. Leandro Navarro Moldes for taking the challenge of advising my PhD work despite my very unusual situation when I joined the program. Without his continuous support it had been materially impossible to do this work. I hope I have honored his confidence in me.

I would also like to thank Pedro García López for disinterestedly helping me to get out of "local minima", take new perspectives and pursue higher goals in my research.

I would publicly recognize the contribution to the many reviewers in conferences and journals who have helped me to improve my research with their constructive critics, with a special mention to Jordi Guitart for doing an exhaustive review of a draft of the thesis, pointing me to many ways to improve the presentation of the ideas.

I am indebted to my colleagues and friends René Brunner, Isaac Chao, Rubén González, Xavier León, Roberto Morales, and Juan Carlos Nieves, for the good moments and good ideas we have shared along this years in conversations over a coffee or a beer, making the pursuing of the PhD not only an academic endeavor, but also an opportunity for personal grow.

Behind a PhD thesis there are many, many hours of work, at any time of the day, any day of the week. I would like to thank Adrián and Nieves for their patience during all those boring weekends and vacations while I needed to concentrate in my work.

A special mention to my friend Natalí Rocha for listening to my little "war stories" and keeping me optimistic and confident when my work was not progressing as I wanted.

I am also thankful to all the people that work in the Department of Computer Architecture for they support, and in particular to Trini for her kindness and resolution to beat bureaucracy.

I wish to thank my family for rising me with values of personal and professional integrity, honesty, hard work and passion for learning.

And finally, I am thankful to the life because I have the privilege of doing what I love the most.

Abstract

Modern service-oriented applications demand the ability to adapt to changing conditions and unexpected situations while maintaining a required QoS. Existing self-adaptation approaches seem inadequate to address this challenge because many of their assumptions are not met on the large-scale, highly dynamic infrastructures where these applications are generally deployed on.

The main motivation of our research is to devise principles that guide the construction of large scale self-adaptive distributed services. We aim to provide sound modeling abstractions based on a clear conceptual background, and their realization as a middleware framework that supports the development of such services.

Taking the inspiration from the concepts of decentralized markets in economics, we propose a solution based on three principles: emergent self-organization, utility driven behavior and model-less adaptation. Based on these principles, we designed Collectives, a middleware framework which provides a comprehensive solution for the diverse adaptation concerns that rise in the development of distributed systems. We tested the soundness and comprehensiveness of the Collectives framework by implementing *e*UDON, a middleware for self-adaptive web services, which we then evaluated extensively by means of a simulation model to analyze its adaptation capabilities in diverse settings.

We found that *e*UDON exhibits the intended properties: it adapts to diverse conditions like peaks in the workload and massive failures, maintaining its QoS and using efficiently the available resources; it is highly scalable and robust; can be implemented on existing services in a non-intrusive way; and do not require any performance model of the services, their workload or the resources they use.

We can conclude that our work proposes a solution for the requirements of self-adaptation in demanding usage scenarios without introducing additional complexity. In that sense, we believe we make a significant contribution towards the development of future generation service-oriented applications.

List of Publications

1. O. Ardaiz, P. Chacin, I. Chao, F. Freitag, and L. Navarro. An architecture for incorporating decentralized economic models in application layer networks. *Multiagent and Grid Systems*, 1(4):287–295, 2005

2. Pablo Chacin, Felix Freitag, Leandro Navarro, Isaac Chao, and Oscar Ardaiz. Integration of decentralized economic models for resource self-management in application layer networks. In Ioannis Stavrakakis and Michael Smirnov, editors, *Autonomic Communication*, volume 3854 of *Lecture Notes in Computer Science*, pages 214–225. Springer Berlin / Heidelberg, 2006

3. Pablo Chacin, Liviu Joita, Björn Schnizler, and Felix Freitag. Flexible architecture for supporting auctions in grids. In *Workshop in Smart Grid Technologies on the Internacional Conference on Autonomic Computing (ICAC 2006)*, 2006

4. P. Chacin and L. Navarro. Collectives: A framework for self-adaptive p2p application. In *Proceedings of the 6th Workshop on Adaptive and Reflexive Middleware (ARM2007)*, New Port Beach, California, USA., November 26 2007

5. Pablo Chacin, Xavier Leon, Rene Brunner, Felix Freitag, and Leandro Navarro. Core services for grid markets. In Thierry Priol and Marco Vanneschi, editors, *From Grids to Service and Pervasive Computing*, pages 205–215. Springer US, 2008

6. Pablo Chacin, Leando Navarro, and Pedro Garcia Lopez. Utility driven service routing over large scale infrastructures. In *Towards a Service-Based Internet. Proceedings of the Thirds European Conference ServiceWave*, volume 6481. Springer Berlin / Heidelberg, 2010

7. Pablo Chacin, Leando Navarro, and Pedro Garcia Lopez. Load balancing on large-scale service infrastructures. Technical Report UPC-DAC-RR-XCSD-2011-1, Polytechnic University of Catalonia, Computer Architecture Deparment. Computer Networks and Distributed Systen Group., 2011

8. Pablo Chacin and Leando Navarro. Utility driven elastic services. In *Proceedings 11th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 6723 of *Lecture Notes in Computer Science*. Jun 6-9 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> Over the last 5 years there has been a developer led revolt against complexity in the middleware.
>
> Paul Maritz, CEO VMware

Modern distributed applications are evolving towards a Service-Oriented Architecture (SOA), dividing their functionality in a set of independent services that offer well defined capabilities [206]. Many large web service providers have leveraged SOA and adopted a "Software as a Service" (SaaS) paradigm, offering application services to third parties that compose them in mashups to create new value added services [185]. Under this paradigm, services are reused and combined, new services are introduced frequently, and usage patterns vary continuously. Additionally, different user segments emerge with different QoS requirements in terms of attributes such as response time, execution cost and security [159]. For example, users can be segmented into service classes (e.g. gold, bronze) with different QoS requirements to separate paid and free access to a service.

The relevance and potential economic impact of this paradigm for Europe is evidenced by the emergence of multiple initiatives around the concept of Internet of Services, like NESSI [1] and Future Internet Assembly[2], which have the specific goals of fostering the research on new technologies and engineering approaches, and promoting its adoption in enterprises as a competitive differentiator.

With the rising popularity and sophistication of this new type of services, providers are required to build infrastructures capable of delivering high volumes of uninterrupted service to their customers. Additionally, those large-scale service-oriented applications frequently address unexpected situations that demand a rapid adaptation of the allocated resources, like flash crowds –that require a quick allocation of additional resources– or massive hardware failures –that require the re-allocation of failed resources.

To facilitate the adaptation of services to these situations, it has become a common practice to deploy them over large-scale non-dedicated infrastructures – e.g. shared clusters – on which servers are dynamically provisioned/decommissioned to services in response to workload variations. This infrastructure can span different administrative domains when servers are located on different clusters or when some of the servers come from a local infrastructure and some others from an external cloud.

[1] http://www.nessi-europe.com
[2] http://www.future-internet.eu/

However, as these infrastructures become larger, more distributed and more heterogeneous, and their usage scenarios more demanding, their manual management and operation become unattainable tasks; moreover, system designers cannot anticipate the adaptation needs at design or even deployment time, as handling unexpected situations may require changing algorithms or even the organization of the system.

This challenging situation and the growing importance of large scale distributed services call for a new approach of system management.

1.1 Scenario Description

The focus of this thesis in on cluster-based locally-distributed web services [43] using non dedicated infrastructures, on which web services are deployed over a set of servers housed together in a single location, interconnected through a high-speed network, presenting a single system image to the outside. However, the approach can also be applied to cloud-based web services, as cloud provider use a similar architecture for their infrastructures [155].

The management of such shared infrastructures must consider two fundamental complexity dimensions: The **Environment Complexity** given by its scale, the dynamics of its configuration, and its openness to new services and usage patterns; and the **Allocation Complexity** as a product of the diversity of users and their requirements, the intricacy of allocation decisions based on multiple parameters, and the existence of different and potentially conflicting QoS objectives for the various services. Figure 1.1 shows how this type of applications – which we term here a Large Scale Distributed Service (LSDS) – compares to other application models with respect this two complexity dimensions.

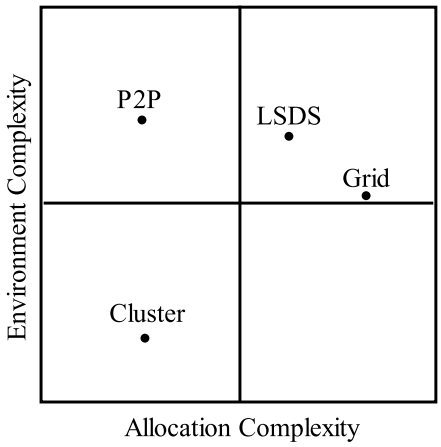


Figure 1.1: LSDS's compared with other common distributed applications classes with respect of their management complexity.

LSDS's share the environment complexity of P2P systems due to their large scale and the changing nature of their infrastructure as instances are activated and deactivated or fail, but the heterogeneity of nodes (servers) is lower and the churn is not that high. This environment complexity is however much higher than the rather static setups of most grid systems. On the other dimension, LSDS's share the management complexity of grid systems with respect of the need of offering multi-attribute QoS, while P2P systems are mostly best-effort and generally have requirements that map to a limited set of attributes like high download bandwidth.

1.2 Problem Statement

In a traditional enterprise infrastructure, adapting to changes in the demand and other unexpected situations would take a long time and require manual intervention, making it impractical. As a consequence, over-provisioning services to handle such situations is the common practice. Unfortunately, over-provisioning is not cost-effective as some services may have high peak-to-mean-to ratios – mostly in the case of exceptional events – and therefore a large portion of the allocated capacity would remain unused for long periods.

Chandra et al. [53] demonstrated that fine-grained multiplexing at short time-scales – in the order of seconds to a few minutes – combined with fractional server allocation leads to substantial performance gains over coarse-grained reallocations and static partitioning. To accomplish this fine grained multiplexing, it is necessary to count with mechanisms to allocate/deallocate servers efficiently, manage configuration changes in a very dynamic environment with a high turn-over of servers, and still allocate requests to service instances maintaining the QoS and using efficiently the allocated resources.

Self-adaptive systems [184] emerge as a promising alternative t handle this management complexity: computing systems capable of modifying their own behavior in response to changes in the operation conditions. However, despite their many potential advantages, the development of self-managed systems is not exempted of challenges.

As noted in [168] traditional closed loop self-adaptation approaches are of limited applicability in the scenarios described above, as they made a set of restrictive assumptions: a) the entire state of the application and all the resources are known/visible to the management component, b) the adaptation ordered by the management component is carried out in full and in a synchronized way, and c) the management component gets full feedback of the results of changes made on the entire system. In contrast, in a large-scale wide-area system getting a global system knowledge is infeasible and coordinating adaptation actions is costly. Additionally, servers may belong to different management domains – different sites in an organization, external providers – with different management objectives.

Moreover, when applying such approaches in non-dedicated infrastructures one additional problem arises. The QoS offered by an instance depends not only on the workload it receives but also on the effect of any other load in the same physical host – for example other service instances – which may not be under the control of the same adaptation process, making the objective of adaptation a moving target. The utilization of a resource isolation mechanism to prevent this interference with the adaptation process would limit its applicability to scenarios where such isolation is not feasible or results impractical. For example, multiple services deployed over the same service container [3].

Additionally, the implementation of some self-adaptation approaches may require the explicit representation and processing of knowledge about the system (components, architecture) and the desired adaptation properties (policies, rules) rising the complexity of the self-adaptation mechanisms and bringing issues of knowledge interoperability among applications and platforms. The complexity of eliciting a model to predict the effect of the adaptation decisions on the QoS may prevent the quick introduction of new services or the adaptation of existing services to sudden changes in the usage patterns or underlaying implementation, two common situations in modern service-oriented applications.

[3] As discussed in chapter 4, even when such performance isolation mechanisms exists, they have some practical limitations.

Finally, the implementation of self-adaptation may result intrusive, imposing programming models, tools or practices to application developers, like the adoption of a component-based development model [76] or the usage of annotations in the source code [132].

Therefore, to meet the challenges of self-adaptive large-scale distributed services, new approaches are needed to address the following fundamental aspects [173] [133] [135] [160]:

- **Conceptual models** defining appropriate abstractions and models for specifying, understanding, controlling, and implementing self-adaptive behaviors;
- **Architectures** to guide the specification and implementation of self-adaptive behaviors of components and their interactions;
- **Middleware infrastructures** that provide the core services required to realize adaptive behaviors in a robust, reliable and scalable manner, in spite of the dynamism and uncertainty of the system;
- **Programming models and frameworks** that support the development of adaptive systems with a clear separation of the adaptation concerns from the application logic.

Additionally, we must consider that:

> "Managing complexity is a key goal of self-adaptive software. If a program must match the complexity of the environment in its own structure it will be very complex indeed! Somehow we need to be able to write software that is **less complex** than the environment in which it is operating yet operate robustly." [142], emphasis added.

The main objective of this thesis is to address these diverse challenges under a comprehensive approach without introducing additional complexity in the systems.

1.3 Requirements

To be effective in the target scenarios and provide the intended benefits, a self-adaptation solution should exhibit the following desirable properties:

R.1 Adaptiveness: Support varying workloads and infrastructure changes

R.2 Application independence: Offer a generic infrastructure that supports multiple services

R.3 Comprehensiveness: Support a broad range of QoS needs

R.4 Efficiency: Achieve good resource utilization

R.5 Endurance: Degrade gracefully under overload

R.6 Flexibility: Accommodate different resource management policies at the node level

R.7 Manageability: Ease of maintain and operate

R.8 Non-intrusiveness: Require a minimal infrastructure modifications

R.9 Reliability: Assign requests despite the unpredictability of the environment

R.10 Resilience: Handle continuous activation/deactivation and failures of instances

R.11 Robustness: Work with incomplete, stale or inconsistent information

R.12 Scalability: Scale to a very large the number of service instances

1.4 Solution Approach

Our approach has been influenced mainly by concepts from decentralized markets. Markets are in essence coordination mechanism available for whatever purposes agents pursue [111]. In particular, they can be used to facilitate agents to maximize utility functions [78] which reflect the goals of the system. Also, from the field of complexity economics, it is known that market mechanisms, when used by adaptive agents, can lead to self-organization in terms of the emergence of stable interaction patterns [17] [197] [136] [205]. More over, bounded-rational agents [192] [18] has been shown to exhibit successful behaviors with highly sophisticated adaptation capabilities using simple, generic models and limited information about the environment [208]. We discuss in more detail these concepts in chapter 2.

Base on that conceptual framework, we have postulated the following key design principles for the self-adaptation of large scale distributed services:

Emergent Self-Organization. Our approach is to use a decentralized, self-organized mechanism in which adaptation decisions are taken independently on each instance, based on local information. This approach has number of advantages. First, the complexity of the adaptation process depends on the size of each instance's neighborhood, instead of the total number of instances, making the system more scalable. Second, it is more robust, as there is no single point of failure. Third, it facilitates the rapid reaction to local situations, like failures or flash crowds. Finally, it allows each instance to manage its own adaptation and thus accommodates the case of multiple administrative domains with different management policies.

Utility Driven. A utility function maps a set of attributes that capture the state of the system and its environment to a single scalar value – conventionally in the $[0, 1]$ range – that measures the relative satisfaction derived from this state. Utility is generally an aggregated function of the benefits, costs and risks associated with a situation (for example, the outcome of an action). In the case of services, utility may consider attributes like the performance (e.g. response time), available resources (e.g. bandwidth), the characteristics of the physical node on which a service runs (e.g. located on a trusted environment or not), execution cost (e.g. energy consumption), and any other relevant attributes.

Utility functions offer a principle basis for rational decision making [78] in the adaptation process. Unlike other approaches that use complex rules over a combination of performance metrics, facilitate comparing configuration alternatives with respect of their fitness to the service's goals [134], making the adaptation process extensible to different definitions of utility.

Model-less. Adaptation does not require either a performance model of the service or a characterization of its workload. This approach offers two advantages which are of particular importance in our scenarios of interest: the effectiveness of the adaptation does not rely of the predictive power of a model, which may be limited by the volatility of the environment; neither does it require eliciting and adjusting modeling parameters to handle new services or workloads, facilitating the provision of a generic infrastructure on which new services can be easily introduced.

It is important to notice that utility functions are not performance models, as they cannot be used to predict future system states, but can only be used to valuate a given state with respect of certain objectives and preferences.

Table 1.1 summarizes how these principles contribute to confront the various challenges we have identified.

Requirement	Emergence	Utility Driven	Model-less
Adaptiveness	✓		
Application independence		✓	✓
Comprehensiveness		✓	✓
Efficiency		✓	
Endurance	✓		
Flexibility			✓
Manageability		✓	✓
Non-intrusiveness			✓
Reliability	✓		
Resilience	✓		
Robustness			✓
Scalability	✓		

Table 1.1: Requirements addressed by the principles that sustain the proposed solution approach.

1.5 Research Questions and Hypotheses

The work presented in this thesis is the result of pursuing the following research questions which have guided our approximation to the different aspects of the problem:

Q.1 Can the economic concepts behind the decentralized markets be generalized as self-adaptation principles for distributed systems without recurring to the market metaphor?

Q.2 What kind of abstractions and programming model are needed to implemented these concepts at the middleware level?

Q.3 To what extent can an approach based on these simple principles offer an *efficient* solution to demanding requirements? Could it adapt to diverse conditions without requiring domain-specific knowledge?

Based on the solution principles described above, we have formulated the following hypotheses to respond to these research questions:

H.1 The principles from economic adaptation can be applied to application level self-adaptation in the form of a model-less adaptation processes driven by utility functions, acting on local information

H.2 Overlays – and epidemic overlays in particular – can be used as a generic self-adaptation middleware which embodies the self-organizing nature of economic systems.

1.6 Methodology

This thesis follows the research methodology presented in figure 1.2. The validation of our hypotheses is done by realizing the principles they propose into a framework, using it to implement a proof of concept application, and studying the resulting properties in diverse scenarios. More concretely, we implemented a middleware to provide self-adaptation capabilities to web services, and evaluated its behavior on a simulated large scale non-dedicated cluster. Even when the proof is limited to this single case, the scenario is complex and demanding enough to serve as an indicative that the approach can be applied to other scenarios that share some key characteristic we have identified and discuss along the thesis.

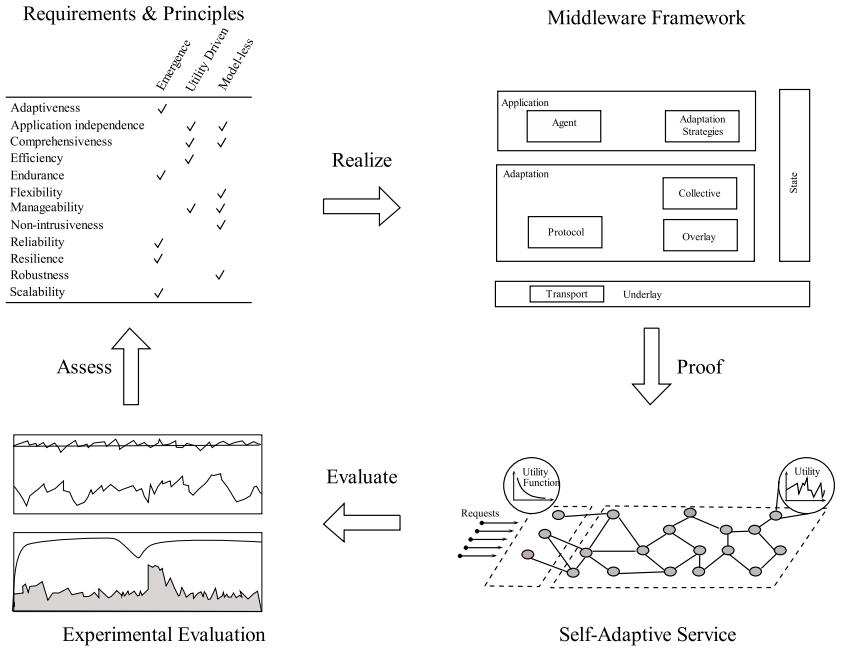


Figure 1.2: Research methodology followed in the thesis to prove the hypotheses.

1.7 Contributions

The major objective of this thesis is to contribute to the understanding on how to build large scale self-adaptive services. In that regard, we are interested in devising architectures more than proposing concrete algorithms. A system's architecture captures its structural characteristics and constraints, from which significant properties can be derived [95]; An architectural approach to the self-adaptation provides an appropriate level of abstraction and generality of concepts and principles developed [139].

This thesis makes the following concrete contributions:

- **Collectives:** a framework for the development of adaptive distributed applications. The main objective of Collectives is to provide both the abstractions needed to encapsulate all the relevant adaptation concerns and the architecture to realize them
- ***e*UDON:** a middleware, based on Collectives, that provides utility driven self-adaptation for service-oriented applications deployed over a large-scale, non-dedicated infrastructures, addressing the need for elasticity and the maintenance of a target QoS in the presence of fluctuations in the load or the available resources.

Both the Collectives and *e*UDON middleare frameworks exist as prototypes. Their evaluation was conducted by simulating the infrastructure – the network and the service instances – while the rest of the core mechanisms and adaptation logic was implemented as code that can be deployed on top of an real infrastructure.

In addition, our research has a number of other contributions that can be applied outside the context of this thesis:

- An exploration of the applicability and limitations of the concepts of decentralized markets in the self-adaptive allocation of resources in distributed services and its realization in a reference architecture (the Grid Market Middleware)
- A characterization of the different aspects involved in the development of adaptive web services and the techniques that can be applied, with emphasis on large scale locally distributed deployments
- The study of alternative epidemic overlays to facilitate the location of service instances deployed over a large scale infrastructure when the QoS of each instance can vary due to fluctuations in the load
- The study of alternative heuristics for load balancing and resource discovery over large scale overlays, when the QoS of nodes have continuous fluctuations.

1.8 The Thesis in Context

Figure 1.3 summarizes this thesis. We address the challenges of self-adaptation in large scale distributed services. Our research follows a multi-disciplinary approach, taking mainly from the complexity economics and self-adaptive systems background. We propose a comprehensive solution in the form of two middleware frameworks, whose design was guided by a set of clear principles derived from the research questions and hypotheses we have proposed.

1.9 Thesis Road Map

This thesis is organized as follows. Chapter 2 introduces the conceptual framework on which this thesis is founded and that support the solution approach we propose. We explore the general concepts of self-adaptation and present the contributions of economic theory to the understanding of the self-adaptation in large scale systems. Chapter 3 presents Collectives, a middleware framework for developing self-adaptive applications based on the concepts of epidemic style emergent self-organization. Collectives addresses the various adaptation concerns

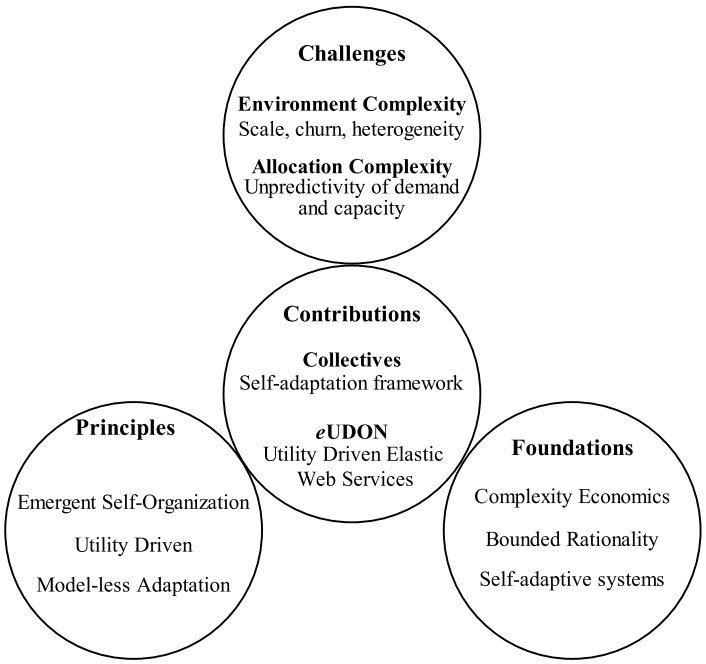


Figure 1.3: The problems addressed, the theoretical background and the contributions.

that are found in the design and implementation of large scale distributed systems. Chapter 4 introduces the problem of self-adaptation in distributed service-oriented applications and presents a logical architecture to understand the diverse aspects that must be considered. Chapter 5 presents *e*UDON a middleware build upon the concepts of Collectives that provides self-adaptation capabilities to large scale distributed services, which address various of the adaptation concerns discussed in chapter 4. Chapter 6 presents the experimental evaluation of *e*UDON, detailing the experimental model and discussing the results under diverse scenarios. Chapter 7 puts the contributions of this thesis in the context of similar works, making a comparison of how our approach differs and improves over other alternatives. We conclude in chapter 8 with a summary of the contributions, a discussion of their relevance in different contexts and a exploration of ideas for future research.

Chapter 2

Background

In this chapter, we provide an overview of the conceptual foundations of this thesis. We start by reviewing the general concept of self-adaptive systems, their objectives, characteristics, challenges and approaches. We then present how economic theory contributes to the understanding of self-adaptation in large scale distributed systems by providing a rich set of concepts and insights on how economic systems can adapt to internal and external changes, and the role that social networks – as a means of self-organization – and rationality play in this process.

2.1 Self-adaptive Systems

Managing large computational infrastructures is a complex endeavor that involves many aspects: a) defining QoS policies for services; b) mapping QoS to resource requirements; c) discovering resources that guarantee an adequate QoS; d) allocating resources according to usage policies; e) monitoring the state of the service; and f) reacting to violations of QoS due to failures or performance degradation, triggering again the resource mapping, discovery and allocation steps.

However, as these infrastructures become larger, more distributed and more heterogeneous, and their usage scenarios more demanding, their manual management and operation become unattainable tasks; moreover, system designers cannot anticipate the adaptation needs at design or even deployment time, as handling unexpected situations may require changing algorithms or even the organization of the system.

Self-adaptive systems has emerged as an alternative to build computing systems capable of modifying their own behavior in response to changes in their operational conditions.

2.1.1 Characterization of Self-Adaptation

Being self-adaptive entails deciding on its own what must be done to keep the behavior of the system stable and within acceptable performance limits, selecting appropriated solutions based on current context and the policies in place [142]. For a system to exhibit self-adaptive behavior, it must possess some attributes which we summarize as follows [152] [193]:

- **Aware:** able to monitor (sense) its operational context as well as its internal state.

- **Adaptive:** able to change its operation (i.e., its configuration, state and functions) to cope with temporal and spatial changes in its operational context.
- **Automatic:** able to self-control its internal functions and operations without any manual intervention or external help.

Self-adaptation manifests in different forms, such as self-optimization, self-configuration, self-healing and self-protecting [152]. Self Configuration is the ability to configure and reconfigure itself under varying and unpredictable conditions. Self Optimization is the ability to detect suboptimal behaviors and optimize itself to improve its execution. Self-Healing is the ability to detect and recover from potential problems and to continue functioning smoothly. Self-Protection is the ability to defend from malicious or accidental attacks and maintain its integrity.

Building self-adaptation capabilities into a system involves diverse considerations [170] [67]. Adaptation can be a fully autonomous process, which is capable to define (and evolve over time) its own goals, or be just a mean to automate the achievement of user defined goals. Adaptation can be limited to a certain predefined application behaviors, or open to new application behaviors introduced at run time. The adaptation can be executed as a continuous optimization process, in an opportunistic way, or on a as-needed basis. The adaptation process can use diverse sources and qualities of information, from purely local to global, from recent to sampled or historical. Self-adaptation can be deemed as a macroscopic property, measured at the global level, or a microscopic property, if it is an attribute of a single entity in the system and its immediate vicinity.

2.1.2 Approaches

To achieve the objectives of self-adaptation, addressing the challenges discussed above, diverse approaches have been proposed:

Biology inspired models start from the realization that living organisms can effectively organize large numbers of unreliable and dynamically changing components (cells, molecules, individuals, etc.) into structures which exhibit properties like robustness to failures of individual components, adaptivity to changing conditions, and the lack of reliance on explicit central coordination. Some of the adaptation patterns found in biological system have been applied to computer systems, like diffusion (equalization), epidemic replication, stigmergy, chemotaxis, morphogen gradients, local inhibition and competition, among others [24] [165]. Biology has also contributed with a family of approaches known as evolutionary computing, which share a common idea: given a population of individuals, the environmental pressure causes natural selection on the best fitting individual rising the fitness of the population over time [82].

Cybernetic models are formalizations of goal directed systems, which incorporate elements from – and have also influenced to – other disciplines like systems theory, information theory, and control theory [1] [198] [39]. This kind of adaptive systems were extensively studied under the concept of ultra-stable systems [19], which consists on two closed loops that maintain a set of critical variables within their operational margins, reacting to short

[1] The terms cybernetics is sometimes abused and then equalized to control theory when the later is a subset of – and a tool for – the former

term and long term disturbances caused by internal or external factors. Many of these concepts are incorporated in the Viable System Model (VSM), a conceptual framework for self-adaptive systems that considers aspects of coordination, control, intelligence, policy and audit [145].

Control Theory models include the classical feed-back or reactive control, on which the current state is observed to detect deviations with respect to a target state, and also predictive control, on which a model of the system is used to predict the future behavior over a prediction horizon [1]. Developing a controller for a system requires the mapping of the particular QoS control problem into a system of feedback loops, developing effective resource models, choosing proper actuators, handling sensor delays, and addressing lead times in effector‘s actions [72] [2].

Agent-Based models use agents, entities capable of perceiving an environment and acting autonomously upon it, as its basic abstraction. There are multiple paradigms to represent agent behaviors like deliberative, reactive, and planing based, among others. This approach leverage the methodologies, architectures and tools for the modeling, design and implementation of systems which covers significant aspects of self-adaptation such as environment awareness, reasoning, organization and coordination [214]. One interesting characteristic of this approach is that it helps in closing the gap between the modeling and the actual implementation of the self-adaptation mechanisms using agent-based engineering methodologies [125]. Examples of this approach are [195] [75] [31].

Social models are based on the concept of social networks formed by individuals and their social connections, contacts, interactions, etc. These models consider how aspects like network topologies and their properties, behavioral patterns, and information dissemination and learning mechanisms impact in individuals preferences, trust, reputation, and other social outcomes [106]. For example [15] explore how concepts related to how socio-economic systems autonomously manage themselves – by making decisions, adapting their structure and behavior, and organizing with other entities in the environment – can be applied in the development of self-adaptive computer systems. In [104] a social tag model is used to allow nodes in a P2P network to coordinate for sharing files. Is important to notice that biological and economics-inspired models have many common elements with social models as some insects exhibit social behavior and obviously markets are social organizations.

Economics based models exploit the insights that economic theory, and more specifically micro-economics, offers to the understanding of how markets work as decentralized coordination mechanism. Based on price signals and competition, markets allow their participants to self-organize to achieve their goals and adapt to changes in the environment and disturbances to individual members [90] [113]. In such systems a state of coordinated actions can emerge through the bartering of self-interested participants, who try to maximize their own utility and choose their actions under incomplete information and bounded rationality [87]. We elaborate more on this model in section 2.2.

2.1.3 Emergent Self-adaptation

Emergence is a phenomenon on which novel system-level organization arises from the local properties and interactions of its constituent elements, increasing in an autonomous way the degree of order [158] [66]. The key elements of this concept are the pass from micro to macro-level properties based on interactions of the components to achieve a coherent state of order.

A close concept to emergence is that of self-organization, defined as "a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control" [66]. Even when similar those two concepts are different. Self-organization does not necessarily implies emergence as a system can be designed to achieve an predefined and predictable organization. On the contrary, emergence – in the case of dynamic systems – implies self-organization.

Emergence can be used as a design strategy for large scale distributed systems [81] and results particularly attractive for self-adaptive systems on which the direct engineering is not feasible due to the complexity of the system and the uncertainty of the environment. When engineering a system by means of emergence, its properties mainly reside in the interactions between components, rather than in the intelligence of individual components [68].

System with emergent properties are not exempt of problems. The engineering process will require new approaches as architecture is no longer dictated by a strict specification of structure, but rather by a set of constrains [97]. Also, some unwanted behaviors, such as oscillations, trashing, abrupt phase changes, or mere chaos can emerge, either due to inherent flaws of algorithms, or caused by faulty elements, or induced by the environment. Therefore, new techniques are required to predict, detect and ameliorate those misbehaviors, as well as for testing proposed designs to detect inherent flaws [162].

2.1.4 Epidemic Style Self-Organization

There has been an increasing interest in epidemic (gossip) algorithms[2] as an underlaying mechanism for supporting self-organized and self-adaptive distributed systems. An epidemic distributed algorithm satisfies, to some extent, the following conditions [32] [61]:

1. Involves periodic, pairwise interactions among participants
2. The information exchanged during these interactions is of bounded size
3. When nodes interact, the state of one or both changes in a way that reflects the state of the other
4. Reliable communication is not assumed
5. The frequency of the interactions is low compared to typical message latencies, so that the protocol costs are negligible
6. There is some form of randomness, generally in the peer selection and in the information dissemination.

The epidemic algorithms have a series of inherent properties [32] [61] that make them particularly suitable to environments with high variability and large scale, namely:

[2] In the rest of this discussion we will give preference to use the term epidemic over gossip and will use interchangeably the terms algorithm and protocol.

- Simplicity. Their behavior and properties emerge from simple rules which are easy to implement.
- Convergent behavior. For example, nodes will agree on a global property within a bounded time [85].
- Emergent structure. Complex stable structures with predictable properties can emerge from the interactions among participants [122].
- Bounded load for participants in terms of processing power, memory and bandwidth consumption.
- Independence from the topology of the underlying network.
- Robust to transient conditions like network or node failures, message lost or information inaccuracies.

One important aspect of these algorithms is that regardless their random nature, some of their global properties are well known. In particular, those regarding the dissemination of information [128] [85]. However, the detailed analysis of the properties of a particular algorithm is in general a complex task, as analytical models are difficult to develop. Therefore, simulation is the more commonly adopted method for evaluating them [27].

Epidemic algorithms are no exempt of some limitations, though [32]. For one hand, they have a limited information carrying capacity due to the bounded information exchange and the (relatively) slow periodicity of the message exchange. Therefore, a high rate of events can quickly exhaust the capacity of the algorithm. Another important limitation of the algorithms is that they are not particularly robust against malicious behaviors and correlated failure patterns, as they depend on randomness and the symmetry of the behavior of the participants.

Epidemic algorithms have shown to be very flexible and have been used many different tasks like information dissemination [11] [83], information aggregation [131] [105], peer discovery and sampling [123] [203], arbitrary topology maintenance [122], building structured overlays [3] [102], coordination [103], maintaining data consistency [71], and clustering or slicing of nodes in a network [163] [182] [204], among others.

Moreover, epidemic algorithms have been proposed as a basic building block and organization paradigm for distributed systems [177] [202] [25] [84]. This comes from the realization that epidemic algorithms are amenable for composition and, due to their simplicity, easy to adapt and extend.

2.2 Economic Self-Adaptation

The proven ability of a free-market economy to adjudicate and satisfy the conflicting needs of millions of human agents makes it a clear prospect as a decentralized organizational principle [86]. Markets are in essence coordination mechanism available for whatever purposes agents pursue [111]. In particular, they can be used to facilitate agents to maximize utility functions [78] which reflect the operational goals of the system, like response time, resource consumption efficiency, among others. The study of economics brings some interesting insights on the behavior of the markets as large scale coordination mechanisms, and the role that rationality plays in them.

2.2.1 Markets as Coordination Devices

Micro-economics, and more specifically, Complexity Economics, has studied how economic organization can be thought as a network, or collection of networks [136] [205]. The nature of these networks influences the outcome of the economic process [21] [88], and at the same time the economic process itself modifies the networks, as the individuals learn not only about the appropriate actions to take, but also about whom they should interact with. The network therefore evolves over time with the evolution of the players and there is a continual feed-back from one to the other. It is then impossible to attribute the resulting structure to one subject only, or to explain for one reason. This resembles the purpose of self-adaptation, which precludes any exogenous control device to achieve an equilibrium state in the face of fluctuations in the environment.

One major inspiration for our work was Hayek's concept of "catallaxy". Hayek and other Neo-Austrian economists understood the market as a decentralized coordination mechanism, as opposed to a centralized command economy where a central entity has global knowledge of the system and commands every entity decisions. For them, the goal of markets is to arrive at a state of coordinated actions, the spontaneous order, which comes into existence through the bartering and communication of the community members – which try to maximize their own utility – with each others and thus achieving a community goal that no single user has planned for [111].

For those economists strongly influenced by Hayek, markets are the set of institutions that surround the use of exchange as a means to achieve human ends. Participants on a market agree to use it as a process for engaging in certain forms of behavior without having to agree on what our ends or goals are. What holds markets together as institutions is this agreement on the means. The market here is nothing more than a communication bus – it is not a central entity of its own, which collects all information and matches market participants using some optimization mechanisms.

One key implication of this perspective is that it suggests that markets are, in Hayek's terms, "ends-independent". Markets have no purpose of their own, other than to serve as a process by which individuals and organizations pursue their own purposes. Markets are available for whatever purpose their participants pursue. Like other social institutions, such as language and the law, markets facilitate social coordination by providing rules and signals that guide actors making choices in a world of uncertainty. From this perspective the idea of aggregating utility is simply meaningless (if not just wrong) since utility is seen as essentially nothing more than a degree of importance attached by a decision making individual to an option, in his comparison of it with other options [137]. Therefore markets are to be judged by the degree to which they coordinate the various plans of individuals and organizations [111].

In this model, a central presumption is "constitutional ignorance ", assuming that it is impossible (and even undesirable) to know all the economic variables needed for a centralized or planned resource allocation. Therefore, the knowledge of participants, or more precisely, the fragmentation of knowledge, becomes a critical issue. The economic problem is about how to achieve the best use of resources, given the limited knowledge of participant agents, for ends whose relative importance only these agents know. Which things are to be considered goods, or how scarce or valuable they are, is something that competition should discover guided by the prices that market offers for the diverse goods and services [108].

2.2.2 Bounded Rationality

Rationality, in the sense that is understood by economics, can be defined as the ability to select the action with the best expected utility (in terms of agent's expectations and preferences), given its available information about the environment [78].

The rational choice capabilities of agents are bounded by diverse factors. From the computational perspective, agents are restricted by the resources available to perform their computations, and by the limits imposed by their architectures regarding what type of knowledge can handle and the adaptation that are capable based on this knowledge [221]. Agents are also limited by the completeness and consistency of the information available, by their capability to process this information, by the inertia to assimilate new information, and by the adequacy of their preferences to match decision to their current environment [78].

An agent's rationality can be provided by three approaches: a) by design, in which the agent designer finds the optimal solution and embeds it the agent; b) by deliberation, in which the agent itself performs a resource constrained explicit deliberation in order to make decisions; and c) by adaptation, in which the agent is equipped with mechanism to adapt its behavior in response to feedback from the environment, so that the quality of its decisions improves over time [221]. These approaches differ is aspects like the resources required to find an optimal solution, the likelihood of finding such solution and the adaptability to changing conditions. In an adaptive approach, the search process conducted by a bounded rational agent depends on several factors [156] such as (1) the memory, or the extent to which an agent is able to encode inferences from history into routines that guide future behavior (2) the form of the search process, or how new information is obtained and the direction in which the agent moves to obtain it, (3) the speed of learning, or the speed at which the agent can adjust its behavior in response to changes in the environment (4) the feedback, or the process by which the environment returns to the agents information required to compare the outcomes of their present strategy with an ideal strategy.

The studies on bounded rationality [192] [18] has shown that agents can achieve successful economic behaviors without requiring sophisticated models or exhaustive information about the environment and still exhibiting highly sophisticated adaptation to changing environment.

Wall [208] characterized a bounded rationality decision making process by the following properties, which define a framework for constructing such process:

1. Information processing tends to be frugal and solutions are simple-minded.
2. New solutions are synthesized by modifying currently implemented one, using a local search.
3. Alternatives are considered one at a time, so search is sequential.
4. The search for a new and better solution is undertaken only when it is observed that the goals are not being satisfied.
5. Search is completed when a satisfying (good enough) solution is implemented.
6. Goals are stated in terms of aspirations, formed by adaptation and learning from experience.
7. Search strategies are developed on the basis of learning and adaptation through experience.

8. The attention the decision maker pays to the environment is the product of learning and adaptation driven by experience.

As was noticed in [176] this model has the advantage that it incorporates the update of aspirations and diverse search models into a simple, yet plausible framework. In addition, decision sequences under this model were shown by Wall to result in a variety of dynamic behaviors.

The possibility of implementing an adaptive behavior with limited processing capabilities and frugal information requirements is appealing because it broadens the applicability of the solution to environments where no accurate information is available and where agents must adapt quickly using limited resources to avoid excessive overhead in the system.

2.2.3 Utility Functions

In economics, utility is a measure of relative satisfaction derived from an outcome. Expected utility has been indicated as a unifying principle for decision-making rules as they allow to express preferences over a set of alternatives [58]. In economics, the concept of preference is fundamental as rational economic agents are expected to chose maximally preferred alternatives [78].

A utility function in the form $U : A^n \rightarrow [0, 1]$ maps a vector $[a0, ..., an]$ of attributes that capture the state of the system and its environment to a single scalar value conventionally in the $[0, 1]$ range. Utility is generally an aggregated function of the benefits, costs and risks associated with a situation (for example, the outcome of an action).

It is important to notice that we follow the convention adopted in the computer science community (see for example [70] [134] [130]) to use the term Utility Function to refer to any preferences function, and not in the more restricted sense used in economics that relates utility functions to preferences in the presence of risk [80].

As mentioned in [134] for self-adaptive systems, utility functions are attractive because of their ability to evaluate attribute points in the state space providing an objective and quantitative basis to compare self-optimization strategies, in order to ensure certain levels of performance, or even assign monetary value to the operation of a system. They thus provide a sound foundation for decision-making, as they can tie those decisions to high-level requirements, concerns and goals.

Utility functions have been applied to multiple domains like the automated management of large, multi-application computing data centers [130], admission control in web applications [45], web services composition [9] [218], workflow execution [146], sensor network optimization [38], and network congestion [126], among others.

In all cases, to make the utility abstraction operational, it is necessary to translate it into an utility function that can capture the definition of utility in the context at hand. Many different functions have been proposed in the literature:

- One commonly used utility function is the Cobb-Douglas formulation $U = T^{\theta}C^{1-\theta}$, which relates the utility to both the execution time T and the cost of execution C.
- In [77], the utility function for a node in a P2P with high churn is defined in terms of its available bandwidth B and the estimated connection time S: $U = B * E(S)$.

- In [134], the utility is a function of the deviation of the service's actual response time RT from a target response time RT_0: $U = (RT_0 - RT)/RT_0$ if $RT < RT_0$, 0 otherwise.

The formulation of the service's utility function for a given service is beyond the scope of this work. This problem of eliciting the utility function has been addressed from multiple perspectives. In economics, the Analytical Hierarchical Process (AHP) [180] is used to establish priorities of the attributes that constitute the utility function.

In the context of self-adaptive systems, in [35] it is elicited in an automatic way by the resource allocator by asking the applications for samples of their utility function at certain critical allocation levels. In [70] it is elicited through statistical correlation over the measurements of relevant quality attributes of the application, as well as characteristics of its runtime environment.

2.2.4 Market based Resource Allocation

Large, complex distributed systems and economic systems share many characteristics: are made of numerous autonomous agents that interact in complex ways, operate in an open, uncertain environment, and pursue goals that may vary over time. They also share the need for efficient, decentralized and scalable coordination mechanisms [22] [113] [197] [17].

This apparent parallelism between economics and distributed systems lead to the idea that a computational system set up along market rules – based on decentralized negotiations and a price system – can allow the system as a whole to adapt to changes in the environment or disturbances to individual members [90] [113].

Several attempts have been made for economic based solutions to very specific resource allocation problems like data replication [44] [140] [194], task allocation and scheduling [37] [210] [207] distributed computing resource allocation [91] [100] [143], and local computing resource allocation [171], to name a few.

It is important to notice that the aim of economics-based models in computer science is fundamentally different from the aim of economic theory. In economics-based models, microeconomic theory is taken as a given and is used as the theory for implementation of computational agents. Whether or not the microeconomic theory actually reflects human behavior is not the critical issue. The important question is instead how microeconomic theory can be utilized for the implementation of successful resource allocation mechanisms in computer systems [217].

Based on these principles, an in order to explore how they could be applied to resource allocation in large scale distributed systems, we proposed a framework called the Grid Market Middleware (GMM) inspired by the concepts of Market Oriented Programming [211] and the Catallactic Information Systems [86]. The GMM addresses the problems of providing a general infrastructure for decentralized markets, a scalable architecture and a high level programming abstractions independent of the market model.

The GMM has been designed under two guiding principles: a) integrate under a common framework the services related to information gathering and dissemination in decentralized market, and b) take advantage of the functionalities provided by overlays to organize a distributed system and allow efficient communications and decentralization.

The architecture of the GMM (see figure 2.1) considers the following layers which progressively abstract

from the technological aspects to the market high level abstractions and the application programming interface:

Market: Market specific participants that implement the market allocation mechanism [114] by engaging in trading interactions. **Participants** are agents (in the general sense, not in the MAS sense) that behave on behalf of resource providers or resource consumers, or meditate between them. Participants are responsible for gathering and evaluating market information and deciding their strategies to sell/buy (e.g. pricing). They can also behave as mediators (broker, arbiter, market makers). The **EERM** provides the capabilities to access grid resources from the grid market. It registers resources in the market and provides information about its availability and relevant performance metrics, which is integrated with the market information. EERM also serves as a gateway to access resources, verifying that the intended access are backed by a previous agreement between the parties (provider and consumer).

Core Maket Services: Services that support the development of participants, enabling them to engage in negotiations for resources. The **Exchange Service** provides a trading infrastructure designed to support different market allocation protocols. The **Market Directory Service** provides a decentralized, market wide registry for participants (providers, consumers) and the resources/services been traded. The **Market Information Service** provides current aggregated information and historical statistics of market indicators, like prices and trade volumes, under publish/subscribe and query interfaces. The **Logging Service** keeps a registry of the transactions for accounting, dispute resolution and security purposes. The **Currency Service** is a distributed banking service which enables users to perform and receive payments for resources usage and sharing using a virtual currency (g-currency. It also serves as an overall regulation system, by restricting users with a limited purchase power leading to price contention during peak demand periods [147].

Distributed Information Services: Generic services that allow an efficient management of information in fully decentralized deployments: processing queries and their responses, filtering messages, aggregating information and ensuring consistency and transactional access to critical data. This layer embodies the ideas of

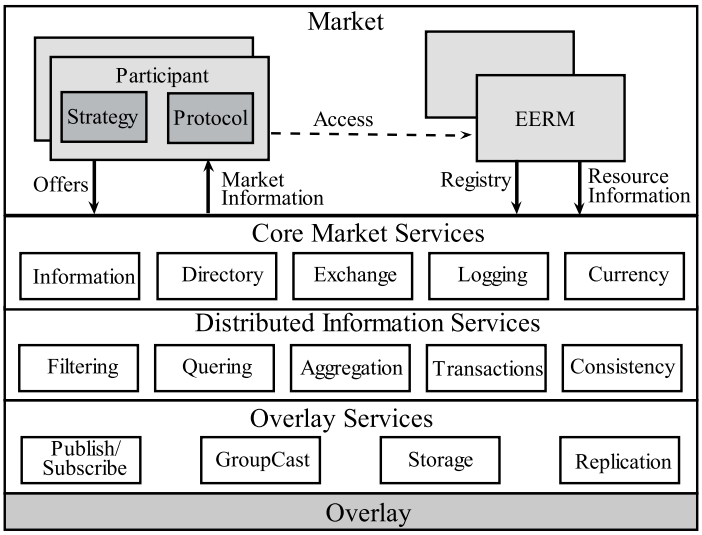


Figure 2.1: Architecture of the GMM

self-organization and information dissemination on decentralized markets.

Overlay Services: Provide sophisticated communication and cooperation mechanisms based on a overlay, like publish/subscribe, group casting, distributed lookup (DHT), and replication. Allows both the static – design time – adaptation to different platforms, as well as dynamic – run time– adaptation to changes in the operational conditions like network topology, churn of participants, surges in workload, etc.

2.2.5 Limitations and an Alternative Approach

Despite the efforts like Market Oriented Programing and the GMM for providing a generic framework to guide the design and implementation of marked-based resource allocation systems, a generic approach for the translation of the adaptation problems to economics principles remains as an open research question.

One of the reasons is that mapping a resource allocation problem to a market model is a non trivial task and is not except of problems [164]. In particular, is not clear the applicability of markets to computational environments, as some of the assumptions commonly made in economic theory may not hold [144] [166]. For example, that prices alone carry all the information necessary for (rational) economic decision making. Additionally, there are practical problems to overcome like how to establish resource pricing and managing virtual currencies whose value may not be well defined outside the computational market [190].

We have therefore opted to avoid the market metaphor and base our approach on the underlaying concepts of utility maximization and bounded rational behavior, within the context of the Catallaxy – i.e. fully decentralized markets operating with incomplete information.

In this sense, we have moved more towards the frontier of social and economics inspired self-adaptation models.

Chapter 3

Collectives

The implementation of self-adaptive systems faces many challenges along its life-cycle, from the modeling to the implementation, deployment and final operation. In this chapter we present Collectives a middleware framework that provides both the modeling concepts needed to encapsulate the relevant adaptation concerns at the proper level of abstraction and the architecture to realize them.

The main contribution of the Collectives is to integrate into a single, comprehensive and extensible model multiple aspects that are subject to adaptation. A clear separation of concerns allows considering individually the different adaptation requirements of the application and use mechanism provided by the proposed framework to adapt the applications to specific requirements.

We propose a concise set of practical design abstractions for understanding, analyzing, and building self-adaptive distributed applications using overlays as the basic organization and communication mechanism. The proposed structuring allows an ordered approach to the development of solutions, by mapping the different concerns to the components of the architecture.

3.0.6 Adaptation Concerns

Current distributed applications have become increasingly complex with respect of their functionalities, environments and infrastructures. Their functionalities are scattered along a large number of nodes which need to collaborate tightly using complex interaction patterns to accomplish their tasks. They must operate in open environments, on which new components, nodes and users can be introduced, migrated or removed at any time [96]. Additionally, when implementing their supporting functionalities, such as searching and data replication, it is difficult to find a single distributed algorithm which behaves appropriately in all scenarios and alternative algorithms must be selected at run time.

As a consequence of all the factors mentioned above, the process of designing and implementing distributed applications must consider the adaptation in different dimensions.

Static-Dynamic Adaptation . The static adaptation considers the configuration and customization of components at implementation or deployment time, while the dynamic adaptation consists in the configuration and tuning of these components at execution time [183]. This adaptation can be achieved either by changing

operational parameters of application components or by changing their composition (arranging elements using different patterns and selecting the elements that participate in them).

Application-Infrastructure Adaptation . At the application level it is necessary to consider adaptation to changes in the policies, user preferences, and usage patterns. At the infrastructure level, to achieve the levels of scalability and efficiency required by large-scale deployments it is indispensable to take advantage of the particularities of the platform by selecting algorithms that exploit them [56] or even consider alternative infrastructures that adapt to different environments.

Local-Global Adaptation . Local adaptation deals with local issues that affects a node and its neighborhood (network congestion, node failure) requiring only local information and little or no coordination of the corrective actions. Global adaptation is necessary to deal with application-wide changes such as variations in the application's workload, which may require global information or coordinated actions.

Proactive-Reactive Adaptation . Applications need to adapt reactively to changes in the environment (e.g. node failures, congestion) but it is also necessary to adapt proactively to achieve application goals.

3.1 Model

The conceptual model of Collectives is based on four key abstractions: Agent, Collective, Protocol and Overlay which are summarized, with their corresponding adaptation concerns, in table 3.1.

Concept	Abstraction	Examples	Adaptation concerns
Agent	Computation component	LRM, object store	Alternative implementations of functions
Collective	Distributed computation	Bag of tasks, DHT	Usage policies, application goals
Overlay	Computation's structure	Random, hypercube, tree	Optimize communications, handle churn
Protocol	Interaction pattern	Broadcast, scatter, gather	Exploit topology, adapt to data flow patterns

Table 3.1: Basic Abstractions and Adaptation Concerns

Agents represent the functionality provided by the application. Each agent offers actions and exposes attributes, which are available to other agents (within a Collective, as seen below). Agents must cooperate to fulfill their functions. For example, agents implementing a distributed storage service must cooperate to search for an item and also to exchange data items for load balancing.

A **Collective** represents an aggregate of agents that interact to fulfill a set of goals and are governed by some policies. The Collective allows agents to invoke actions on other agents and to obtain a global view of the state of the collective. The Collective can also trigger actions in its constituent agents to fulfill the policies in place, like the load balancing in a distributed storage system. Moving this proactivity out from individual agents to the Collective makes the resulting applications more flexible.

The Collective maintains a global view of its state by inquiring the attributes of each agent and aggregating them using diverse aggregation protocols, which offer different characteristics of this global view in terms of consistency and accuracy.

Agents that participate in a Collective are organized in an **Overlay**, an application driven self-organizing and self-adaptive communication infrastructure. It abstracts from underlying network infrastructure (the underlay [167]) and allows the agents to engage in complex interactions, adapting its topology to the needs of the interaction patterns and the network conditions. The overlay can be adapted by gathering network and application level attributes from other agents and filtering them based on these attributes.

The actions initiated by an agent and the information collected by a Collective are propagated by means of **Protocols**, which control how the agents interact. First they define the scope of one agent's neighborhood within the overlay (e.g. all within a "radius", or a fixed number of randomly selected agents). Second, control how the agents in the neighborhood are visited to perform a function (e.g. all the neighbors, or the first one who responds). Finally, they can also filter the view to those agents that meet certain conditions.

The relationship among these concepts is shown in figure 3.1

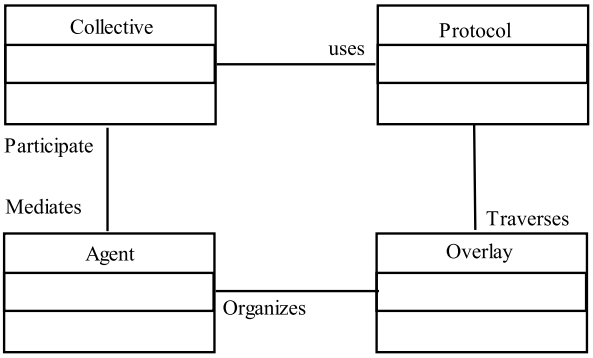


Figure 3.1: Conceptual Mode of Collectives

The model of Collectives can be analyzed along four axes, which separate the main adaptation concerns in the design of distributed self-adaptive applications, as can be seen in figure-3.2.

Structural-Collaboration. Agent and Protocol present the structural abstractions that describe the components of the system, while Collective and Overlay abstract the collaborations on which these components are involved. Structural elements adapt by changing their parameters, while collaboration elements adapt by changing their composition (selecting structural elements or arranging them using different patterns) [110].

Application-Network. Agent and Collective deal with the application-specific issues (e.g. policies), while Protocol and the Overlay deal with the network foundation (e.g. optimize message routing).

Local-Global. the Agent and Protocol deal with to local adaptation issues (e.g. minimizing message traffic), while the Collective and the Overlay deal with global adaptation (e.g. network-wide topological changes and variations in the application's workload).

Proactive-Reactive. Protocols and Overlays adapt reactively to events (node failures, congestion) while Agents and Collective adapt proactively to achieve application goals.

It is important to notice, however, that such clear-cut separation along the axes is not always possible, as cross-axis adaptations are frequently required.

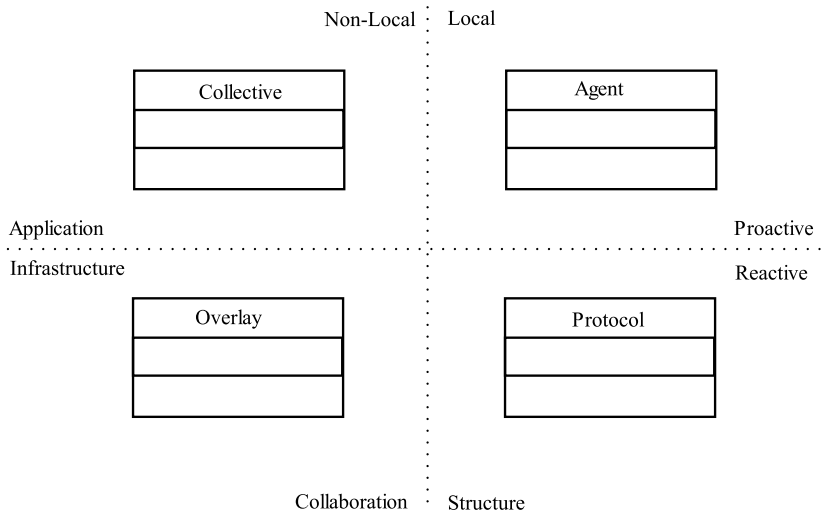


Figure 3.2: Separation of concerns in Collectives

3.2 Architecture

In this section we present how the concepts of Collectives are realized in an architecture for self-adaptive distributed applications. This architecture is organized, as shown in figure 3.3, in three layers: Underlay, Adaptation and Application, which can communicate by means of a shared state that allows a cross-layer cooperation.

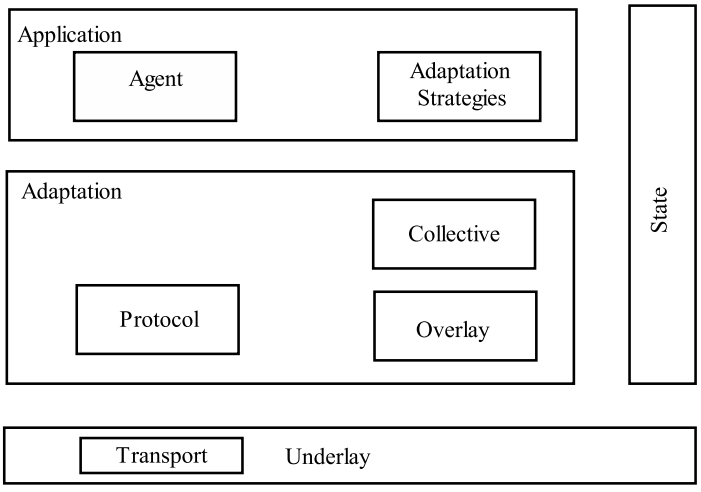


Figure 3.3: General Architecture of Collectives

The Underlay [167, 187] provides application-independent network capabilities, like finding adjacent nodes, delivering messages to a given node and also proving performance metrics such as latency and distance to other

nodes. The **Adaptation** layer provides the mechanisms for adaptation. The **Application** layer provides the application-specific knowledge to adapt the behavior of the Collective. The local **State** is formed by a set of variables – usually mapped to agent's attributes –that allows cross-layer adaptation. The components of these layers are detailed in the next sections.

3.2.1 Overlay

The function of the overlay is to self-organize agents into an application level topology and offer a routing mechanism to propagate messages efficiently over such topology. It serves as a communication substrate which can be used by application protocols to implement complex communication models.

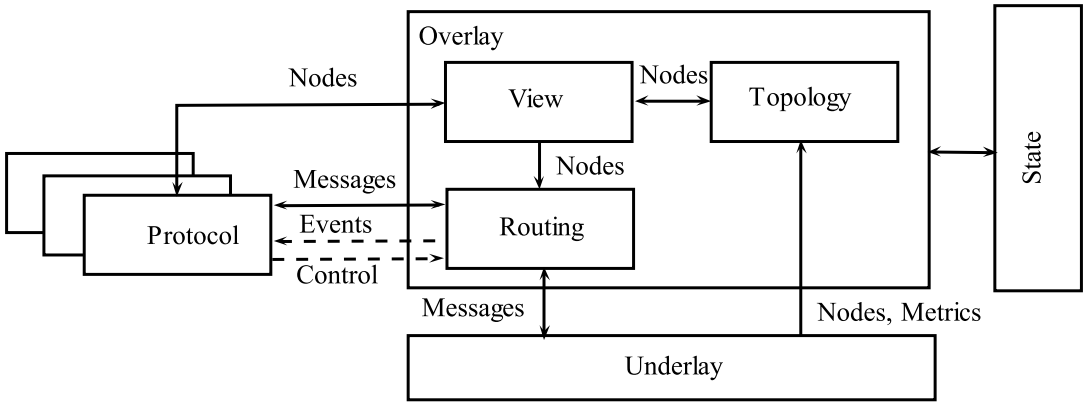


Figure 3.4: Overlay Architecture

The Overlay allows the organization of agents by selecting those which better fit an application-specific selection criteria (e.g. physical distance, closeness of their logical ids, semantic similitude) in order to optimize the efficiency of the communication, also under an application-specific metric (e.g. number of hops, response time, search hit ratio). It builds on the concept of emergent overlays [93] [122] on which nodes self-organize by means of simple rules in response to local information, without a predefined globals structure. More specifically, the overlay uses epidemic algorithms to construct the topology and disseminate information about the nodes to leverage their scalability, robustness, and resilience [32] [85]. Collectives uses push style algorithms because they have a fast initial propagation rate [128], a desirable property when the information is used locally and a system-wide propagation is not needed. Additionally, they are simple to implement using a lightweight communication protocol like UDP, not requiring synchronization between nodes.

Figure 3.5 shows a generic epidemic algorithm. Periodically each node selects a subset of its neighbors (the exchange set) and sends a message with its local view (the neighbor set) and its own current state. When a node receives this message, merges it with its current neighbor set and selects a subset as the new neighbor set. The different epidemic algorithms proposed in the literature differ basically in how they select the neighbors to be contacted, and how they merge the information received (see [124] for a study of different alternatives and their properties).

As it has been shown in [61] [84] [177] [202] epidemic protocols can be used as basic building blocks for implementing different functionalities in distributed systems. This comes from the realization that epidemic algorithms are amenable for composition and easily adaptable and extendable. Collectives takes advantage of

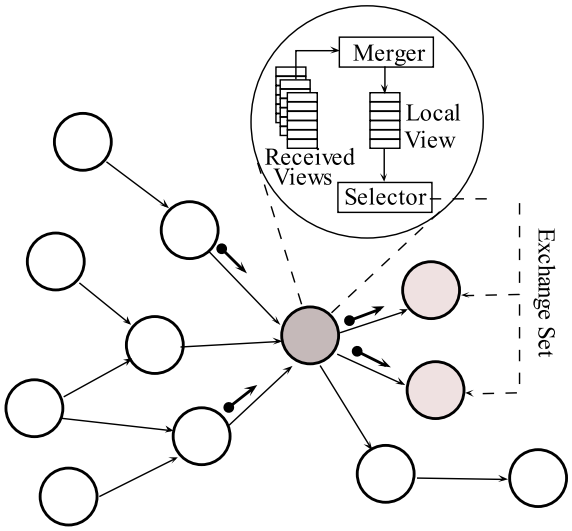


Figure 3.5: A generic epidemic overlay maintenance process.

these characteristics by using a highly modular epidemic overlay as the basic structuring mechanism.

Collectives allow the application to tailor the epidemic overlay construction by specifying the functions used to select the exchange set, the protocol used to exchange the view, and the function used to merge the local view with those received from other nodes.

The composition of epidemic algorithms in Collectives follows the principles outlined in [154] regarding the potential synergies between overlays, but adapted as composition patterns shown in figure 3.6. The composition can be done either horizontally or vertically and can consider different capabilities of each overlay. In the horizontal composition, overlays use each other's capabilities or share a common capability, while in the vertical composition one overlay uses the other overlay's capabilities, establishing a hierarchy. The composition can consider communication and state capabilities. In the case of the communications capabilities, in the vertical composition one overlay uses the other as its communication substrate, while in the horizontal composition both overlays share the same communication allowing an optimization of the communications like those proposed in [201]. With respect to the state composition, in the vertical composition one overlay have access to the state of the other overlay – for example, its routing tables – and receives notifications of changes on that state. In [163] this approach is used to create per-application slices of a large overlay. In the horizontal composition, both overlays cooperate to maintain a shared state, as in the Synergy overlay [141].

3.2.2 Routing and Protocols

The **Router** forwards messages to a destination over the topology. A routing destination is defined as a set of constraints on the attributes of a node that must be satisfied to process a message, as proposed in [219]. The objective of the routing process is to deliver messages in an efficient way, selecting at each step the best path considering the available information about the neighbors and the constrains of the destination.

Collectives provides a modular multi-hop router based on the three functions shown in figure 3.7: Admission Control, Routing Algorithm and Ranking. These functions capture the main routing decisions on which application-specific logic can be used to adapt the routing process to the application's needs.

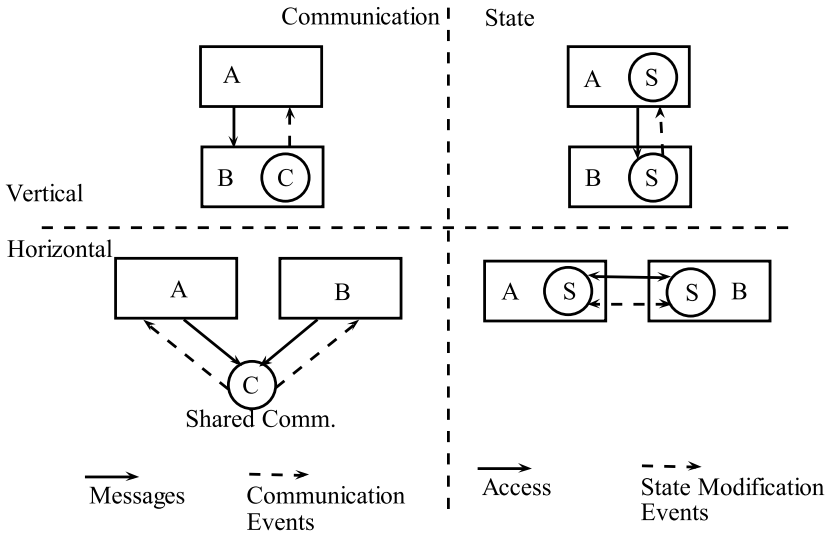


Figure 3.6: Different approaches for overlay integration (adapted from [154]).

On each node, the **Admission Control** function is used to determine if the node will process the message. If so, the message is delivered. Otherwise, it is forwarded to the next hop. The admission control function applies policies to, for example, prevent overloading.

The **Ranking Function** orders (and potentially filters) the nodes from the local view according to their attributes and the destination. For example, considering the distance to the destination, the load of the node to achieve load balancing, or the past experience on routing through each neighbor [115] [219].

The **Routing Algorithm** selects the next hop based on the ranking. Examples are: a probabilistic selection proportional to the ranking, a greedy selection of the top ranked node, a weighted round-robin considering each node's ranking, among others.

The routing process detects and suppresses duplicated messages produced by loops, recovers from transient failures and can attempt alternative routes. In this way the protocols can handle messages without considering these details.

The router also supports multicast routing, on which the message continues to be propagated even if delivered to a node. The multicast propagation can be controlled by means of the mechanisms offered by the RouteObserver interface as discussed below.

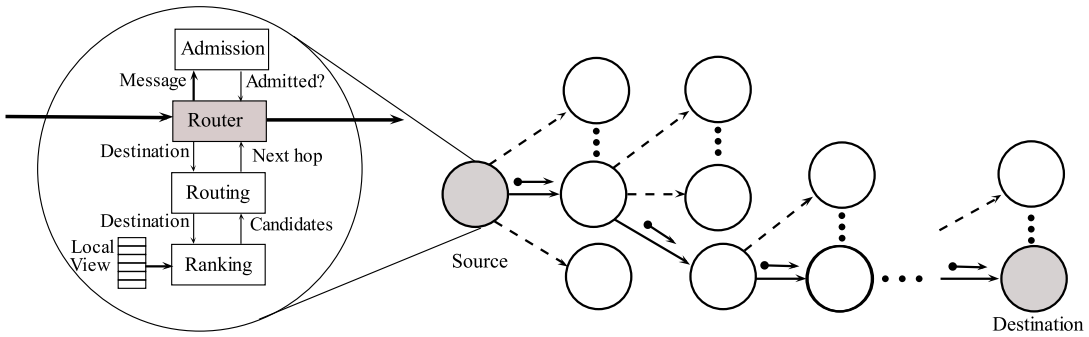


Figure 3.7: Routing

The **Protocols** implement application-specific communication patterns – which carry the requests and

information among agents that participate in a collective– using the routing service. On each step, the protocol receives the incoming message, can trigger actions in the Collective, forward any response back to originator and decide to modify the destination of the message.

Protocols have access to the overlay's topology by means of a **View**, which filters it according to some attributes. Views also allow maintaining protocol-specific information about nodes. One or more protocols can share the same view to allow a protocol to add some attributes used by other protocol. For instance, a protocol can be used to find nodes with a certain attribute and other protocols can then deliver messages only to those nodes. The protocols can also use views to propose candidates to be included in the topology [1].

The interface between the application protocols and the router follows an model similar to that of the common API for structured peer-to-peer overlays [63], even when Collective's overlay is non-structured. In particular, the Router offers the interface RouteObserver to handle the main events during the routing process and allows the protocol to control it:

- When the routing process starts at the source node. The protocol can modify the destination.
- When a message is to be delivered to a node. The protocol can reject the message.
- When a message is dropped because no suitable destination was reached and the TTL was exhausted. The protocol can, for example, switch to a different routing algorithm or modify the destination.
- When no suitable next hop is found. The protocol can retry later or even change the destination.

Multiple route observers can be combined in a chain to create complex logic from basic building blocks. These events can also be used to gather protocol-specific information about other nodes. For example, a protocol used for searching can use the reception of a response to update its view with statistics of the number of responses received from each node. This information can later be used by the ranking function to deliver queries to nodes based on their past performance [2].

Additionally, the router offers extensive instrumentation points to gather statistics about messages routed, forwarded, delivered, failed, and dropped, which are accessible to the protocols.

3.2.3 Collective and Agents

The architecture of a Collective, as presented in figure 3.8, is formed by a set of Actions and the Adaptation Manager.

The main **Application** component is the Agent. Agents must implement an interface with two methods: *visit* and *inquire*. *Visit* is used by the Collective to execute an action and receive a response. *Inquire* is used to retrieve agent's attributes.

The **Collective** offers two methods to agents. The *visit* method allows an agent to invoke an action on other agents belonging to the collective and the *inquire* method allows agents to retrieve a global attribute of the collective. Those global attributes are maintained by the Collective using aggregation protocols.

The **Actions** are the interface to the application-specific functions provided by Agents. Actions can act only locally or be propagated to other agents in the Collective. They can also read and modify the local state.

[1] This is actually how the topologies are maintained by the overlay's maintenance algorithm.

[2] see [115] for a discussion on several of similar heuristics and their impact on searching over unstructured overlays.

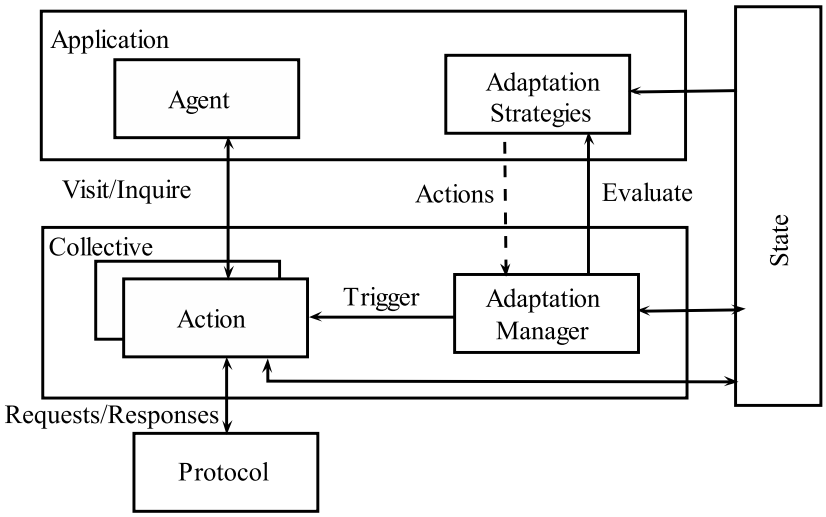


Figure 3.8: Architecture of a Collective

The actions are triggered either by agents, when they request an action to the collective, or by the Collective's **Adaptation Manager** following the application provided **Adaptation Strategies**. The description of the Actions include the binding of the action's parameters to state attributes, the function that must be invoked on the agent, and the protocol and destination to be used to propagate to other nodes, if any. Any of these elements can be changed dynamically at run-time.

3.2.4 Adaptation Strategies, Rules and Actions

The Adaptation Strategies provide application-specific adaptation logic based on Rules and Actions. Rules are functions that returns a value in the [0 : 1.0] interval. Actions are triggered depending on the value of the associated rule(s). Having a real valuation for rules allows not only simple true/false conditions to determine if an action must be executed, but also more complex probabilistic or fuzzy conditions. Some Rules provided by the Collectives framework are given in the table 3.2.

Rule		Returned value
Simple	Ratio	Ratio of an state attribute with respect of a maximum value
	Random	A random value following given probability distribution
Composite	Weighted	Weighted sum of the individual rule's values

Table 3.2: Examples of Adaptation rules.

The **Adaptation Manager** evaluates the strategies when certain conditions on the local state occur (e.g. periodically, or when a state variable changes). Strategies trigger the execution of corresponding adaptation action(s). Some examples provided by the framework are shown in table 3.3. In simple strategies, actions are triggered depending of the rules's valuation. Composite strategies are formed by multiple actions, each one with an associated rule. One or more actions are triggered depending on if corresponding rule's valuations. It is clear that some strategies can be implemented combining others but are offered as separated types for simplicity. For

example, the random composite strategy can be implemented as a probabilistic composite strategy on which each action has an uniform random rule. As part of the future work, we are planning to implement some other commonly used adaptation strategies, like reinforcement learning [196], which presently requires a considerable development effort.

Strategy		Action Triggered
Simple	Probabilistic	With a probability proportional to the rule's valuation
	Threshold	If rule's valuation above(below) a threshold
Composite	Greedy	The action with the highest valuation
	Probabilistic	One action with a probability proportional to its rule's valuation
	Threshold	Every action above (below) a threshold
	Random	A Randomly chosen action

Table 3.3: Some examples the of diverse adaptation strategies that can be implemented using the Collectives framework's adaptation rules and actions.

3.3 Discussion

Raising the level of abstraction when designing a software architecture brings many significant benefits. Reasoning on models brings the focus to the global aspects of the architecture instead of the particularities of specific cases. This is of particular importance when one of the adaptation aspects to consider is the utilization of alternative algorithms for the different components of the system; in this case one can be concerned about the properties of the solution despite the specific algorithms it uses. Moreover, the use of a conceptual architecture helps the designers to better understand and compare alternative designs with respect to the aspects covered by the model – in our case, adaptation concerns.

However, a model should satisfy some key requirements to be practical [186]. It must be understandable; inexpensive to develop with respect of the complexity of the system it models; and executable, in the sense it should lead to an implementation without a significant translation effort to pass from the abstract concepts to the implementation artifacts. Collectives addresses all these requirements by using abstractions that easily capture the problem domain – adaptation is distributed systems – and are provided as implementation artifacts by the middleware framework.

Another important advantage of a clear encapsulation of the adaptation concerns is that it facilitates the reutilization and composition of basic elements to create more complex behaviors. This can be more clearly appreciated in chapter 5 when we discuss the implementation of a middleware for self-adaptive services by composing these basic building blocks.

Chapter 4

Adaptive Web Services

In this chapter we introduce the basic concepts of service-oriented architectures. We discuss their characteristics, describe the scenarios of interest for this thesis, and propose a model to understand the functional components that participate in providing self-adaptation capabilities to such systems. This model offers a point of reference to understand the concerns, scope and limitations of the middleware infrastructure for self-adaptation proposed in chapter 5.

4.1 Service-Oriented Architecture

Modern distributed applications are evolving towards a Service-Oriented Architecture (SOA), dividing their functionality in a set of independent services, each of them offering a well defined capability. More formally, the W3C consortium [206] defines a SOA as:

> A set of components which can be invoked, and whose interface descriptions can be published and discovered

Where a service is defined as:

> [A]n abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities. To be used, a service must be realized by a concrete provider agent.

The term service covers a wide range of concepts including physical resources (computation, communication, and storage), informational services (databases, archives, instruments), individuals (people and the knowledge they represent), capabilities (software packages and supporting services) [92] [65].

Many large web service providers have leveraged SOA and adopted a "Software as a Service" (SaaS) paradigm, offering application services to third parties which can be composed to create new value added services [185]. Under this model services are reused and combined, new services are introduced frequently, and usage patterns vary continuously. This paradigm has been further extended to offer application platform services to support the deployment of third party services (Platform as a Service or PaaS) and raw computational resources (Infrastructure as a Service or IaaS).

In the context of this thesis, we adopt the following formalization – adapted from [120] –of SOA and the problem of QoS-aware request allocation:

Node: A specific type of capability which can run an instance of a Service. We denote as $\mathcal{N} = \{n_1; n_2 \dots\}$ the set of nodes of a SOA based system.

Service: A software functionality available on-request via a network. Each service implements a particular set of capabilities. We denote as $\mathcal{S} = \{s_1, s_2 \dots\}$ the set of services of a SOA based system.

Service Instance: An activation of a service in a node that enables it to process service requests. We denote as $I_s \subset \mathcal{N}$ the set instances of the service s (nodes on which the service has active instances).

Service Consumer: An entity using a particular service. We denote as $C(s)$ as the set of customers for service s

Request: An atomic unit of work, issued by a service consumer, to be processed by a service.

Workload: A series of service requests generated by a service consumer. For each consumer c_i, we denote w_i as the workload and $W_s = \{w_i\}, i\ C_s$ as the workload for a service s.

Quality of service (QoS): A non-negative real-valued quantity specifying the expected execution attributes of a request, such as response time, execution cost, security and others [159]. We denote as $Q_s(c)$ the expected QoS of the customer c for the service s.

Utility: A non-negative real-valued quantity specifying how well a request can be executed in a node. We denote as $U_s(n)$ the utility of the node n for a request of the service s.

Overlay: An undirected graph $\mathcal{O} = \{V, E\}$ where $V \subset \mathcal{N}$ is the set vertexes and E is the set of edges. An edge $e = (n_i, n_j), n_i, n_j \in V$ defines a non symmetric and non transitive relationship between n_i and n_j. In an overlay, we denote the neighborhood of $N(n)$ as the subset $(n, n_i) of E, n_i \neq n$.

4.2 Model

Internet services can be seen as a stream of requests coming from clients through the Internet, that are received by a site, processed by a service instance using resources provided by a server, and returned to the clients upon completion [33].

The focus of this thesis in on cluster-based locally-distributed web services [43] using non dedicated infrastructures, on which web services are deployed over a set of machines housed together in a single location, interconnected through a high-speed network, and presenting a single system image to the outside. However, this approach can also be applied to cloud based web services, as they follow a similar architecture [155].

A model for locally replicated web services running on non-dedicated servers is shown in figure 4.1. Every server – or physical node – supports one or more instances of different services. The resources provided by the node – memory and CPU, network bandwidth, disk – are shared by these instances. Each instance processes requests using a dispatching discipline. In this thesis we assume that each service instance dispatches requests using a processor sharing discipline. This model fits well for web servers like Apache, a well-known and widely used multi-threaded web server, and is amenable to analytical evaluation using a $M/G/1/k*PS$ queuing system [40].

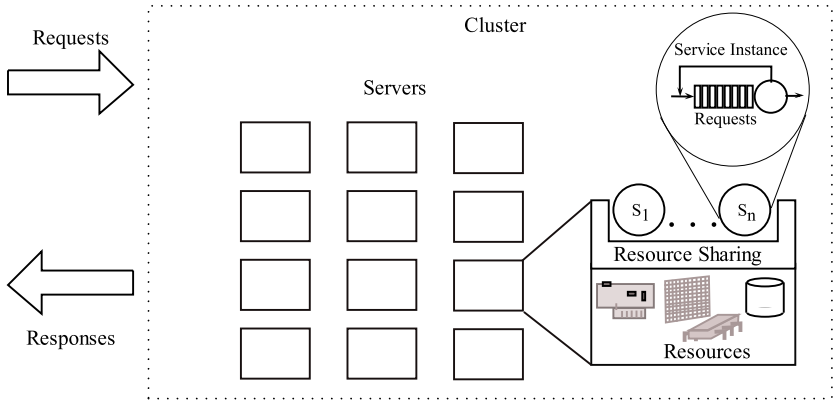


Figure 4.1: Model for a Web Service running on non-dedicated servers.

We focus on Internet services in which each of its incoming request carries with it a specific amount of computation to be performed which must be completed before the request is fully returned to the client. It is important to notice that not all the existing online services fall into this definition of web services. For example, asynchronous systems which would receive (and acknowledge) the requests but that continues processing and sending responses incrementally. Examples of such services are video streaming and push style notifications (e.g. a chat).

An important characteristic of this model is that service instances must execute multiple independent requests which have similar execution characteristics and QoS requirements. As a consequence, each instance is able to estimate the QoS it can offer considering only its current execution state. A contrasting case is the execution of jobs in a grid where every job may have different requirements, like the number of processors, memory, total execution time or the access to certain dataset replicas. In that scenario, an instance could not assess the QoS it can offer to a request until it receives it and evaluates the request's requirements.

One additional assumption we made is that we consider fine grained services, on which individual requests represent a small fraction of the overall workload and that the service level objectives allow for a fraction of those requests to under-perform (e.g. it is expected that %95 of request to be under a certain response time, so the remaining %5 percent can miss this goal). This case contrasts with the scenario of scheduling a parallel job on which each process represents a substantial amount of work and the delay of one process may affect the performance of the whole job (due to task dependencies) [189].

Finally, it is important to notice that in our work we concentrate in the web application layer and assume that the data access, including consistency requirements, are handled by a separated data layer as proposed in modern highly scalable web architectures [138]. Moreover, we assume services are stateless in the sense that

even when session affinity can be desirable, a request can be attended by any service instance. This property can be achieved by using a separate distributed cache for session handling. Despite this assumption, we propose alternatives – discussed in section 5.8 to handle situations when session affinity is mandatory.

For our study, two aspects of the environment are the most relevant: scale and dynamism. The *scale* determines how many servers are considered. We differentiate low (several tens to one hundred), medium (a few hundreds) or large (several hundreds to thousands) scale. The *dynamism* measures to what extent the configuration – number and characteristics of nodes – is mostly static (low) or changes frequently (high).

The workload can be characterized in terms of its *arrival rate*, that influences how frequently dispatching decisions are made, and the *granularity* of the requests, that defines how much the processing of each individual request can affect a server's state. These two characteristics influence how much the system state can vary between information updates.

Table 4.1 compares various classes of applications found in the literature (job scheduling/Grid, traditional web servers and P2P) to our scenario of interest – which we identify as Large-Scale Distributed Services (LSDS) – using the characteristics discussed above. It is clear that LSDS' share characteristics with those other scenarios and therefore can benefit from the approaches used in their contexts, but also introduce new challenges due the scale and dynamism of the environment, requiring new approaches. This need is one of the main motivations of the present work.

	Grid	Web	P2P	LSDS
Scale	medium	Low/Medium	Large/Very Large	Large
Dynamism	Medium[1]	Low	Vey High	High
Arrival rate	Low	High	High	High
Granularity	Large/medium[2]	Small	Medium[3]	Small

Table 4.1: Comparison of request routing scenarios

4.3 Conceptual Architecture

The management of web services that face changing conditions in their environment and workload is a well studied problem. Even when some good surveys on the subject exists (see for example [43] [101]) they are more focused on describing and comparing existing systems rather than in guiding the design of new solutions. Therefore, to better understand the requirements of adaptation for web services and put our approach in perspective with respect of alternative approaches, we have developed a conceptual architecture for adaptive web services.

We have separated the main design concerns into three components that form this architecture as shown in figure 4.2. The **Service Sizing** decides the number and location of service instances needed to satisfy the allowed workload with the desired QoS. The objective of the **Request Allocation** is to route requests to a service instance that can process it with the desired QoS, while preventing overloading of the instance. Finally,

[1] This Dynamism comes mostly from the consideration of heterogeneous or non-dedicated serves, but configurations are generally considered stable at short term.

[2] [189] considers medium sized service requests, like jobs of short duration.

[3] Mostly for file sharing and streaming, the two more common P2P usage scenarios.

the **Monitoring** component provides the aggregated information for the other two functions to work properly. In the following section we explore each of the components and discuss relevant work.

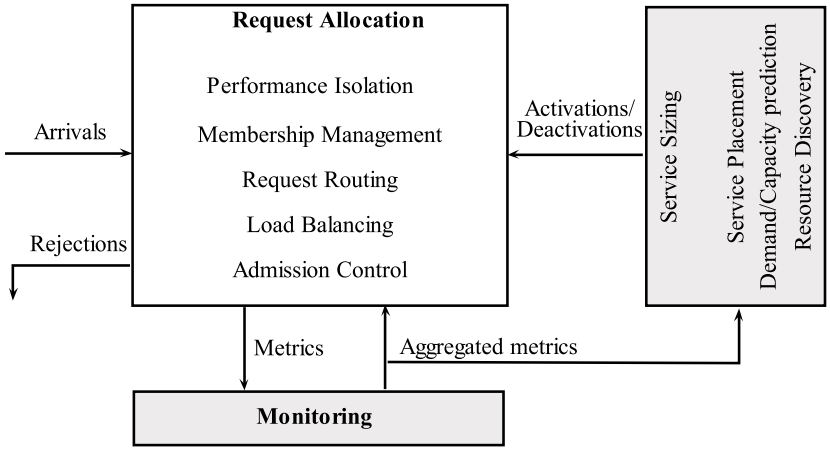


Figure 4.2: A conceptual architecture that identifies the main functional components involved in adaptive web services.

4.3.1 Service Placement

To adapt to changes in the demand or to eventual failures of nodes, it is necessary to periodically decide the number of active instances for each service and their placement over the pool of shared servers. One approach is to consider the problem as a global optimization problem and solve it either centralized or decentralized way, but considering the maximization of a global objective function. For example, in [129] the dynamic placement of the instances of multiple applications on a set of server machines is formulated as a two dimensional packaging problem. However solving this kind of optimization problems has a high computational complexity, severely limiting its scalability.

VioCluster [179] uses a decentralized approach in which service domains brokers negotiate the number of virtual machine instances each domain will borrow/lend from each other, seeking to maximize their throughput.

A totally different approach is to rely only in local information for decision making. In [157] the application placement is modeled using game theory as a minority game on which agents representing the applications decide autonomously on which server to run. In [4] each server autonomously decide which applications to run based on information from its neighbors – about resource usage and requests rate – and a utility function that defines the utility of serving each service. Similarly, in [89] each service instance of a service deployed over an overlay uses a heuristic based on local information from neighbors to decide when to migrate part of the current load to other nodes.

4.3.2 Resource Discovery

The service placement function requires the location of resources which can provide an adequate quality of service (e.g. that have sufficient capacity) to be considered for the placement of an instance. In centralized or hierarchical information systems the resources are registered in a predefined set of nodes and their state is periodically updated. Example of this approach are grid information systems [62]. For very large scale systems

on which resources change frequently, this approach may not work well due to the overhead of continuously updating global information.

A different approach has been to use structured overlays for resource allocation to leverage their performance guarantees (see for example [107] [112] [28]). The general idea is to construct a multidimensional DHT that allows queries over multiple attributes. Two major limitation of this approach are that a) in general, requires that the attributes (and in some cases the range of values for each attribute) to be fixed and known in advance; and b) the maintenance cost makes them unsuitable for frequently changing attributes.

Non-structured approaches overcome these limitations to the expense of only having best-effort performance and need to prevent overflowing the entire overlay with queries. In [115] diverse query dissemination heuristics are proposed to improve the accuracy and latency of queries. Resource slicing or clustering approaches [182] [163] limit the search to a subgroup of potential nodes. Kelips [102] proposes an hybrid epidemic-clustering approach to obtain $O(1)$ look-up time, but its applicability to highly variable environments has not been explored.

4.3.3 Demand and Capacity Prediction

One fundamental problem of service placement is the forecasting of the service demand to estimate future resource demand. In the case of non-dedicated servers – that is, servers whose load is not completely under the control of the service placement mechanism – it is also necessary to forecast the server utilization to predict future system performance and guide the adaptation process.

Workload prediction: Tries to characterize the workload seen by a service by two properties: the request arrival process and the service demand distribution. The general approach is using time series analysis techniques. In [55] the arrival rate is predicted using an autoregressive model based on the last observed arrival rate (an $AR(1)$ model), while th service demand is characterized by its probability distribution derived from the histogram of pass observed service times. In [116] web server access data is viewed as a realization of a set of underlying time-varying (non-stationary) stochastic processes, and auto-regressive time-series analysis to obtain the key properties of this set of processes. A different approach is used in [109] where the non-stationary behavior of the mean of requests per second is characterized considering the influences of time-of-day, day-of-week, and month as well as time serial correlations, which can be used to predict demand several minutes or hours ahead.

Server utilization prediction: Takes as input recent measurements of resource utilization and tries to predict future utilization. In [74] multiple linear and non-linear regression models for load prediction are evaluated, including autoregressive, moving average, autoregressive moving average (ARMA), autoregressive integrated moving average (ARIMA), and autoregressive fractionally integrated moving average (ARFIMA) models. Their results show that the simple autoregressive model has a good predictive power and low overhead, while more complex liner models are expensive to fit and hence difficult to use in a dynamic or real-time setting. The Network Weather Service [213] uses an adaptive, non-parametric approach, applying a set of one-step-ahead forecasting models and dynamically choosing the one that has been most accurate over the recent set of measurements. In [216] simple predictors that track on recent trends are proposed and are shown to

outperform other regression based approaches. In [13] a two-step approach is proposed that first evaluates the load trend through a load tracker function, and then applies the load predictor to the load trend results, instead of working on direct resource measures. A different approach is used in [174] where diverse data mining and machine learning techniques for short time forecasting are evaluated, finding that methods based on Bayesian network classifiers and multivariate regression perform better (on average) for a variety of tasks over univariate auto-regression methods.

4.3.4 Performance Isolation

In non-dedicated infrastructures services hosted on the same physical node may interfere with each other's performance. To prevent this, some degree of performance isolation can be achieved through scheduling policies and/or resource partitioning mechanisms[1].

Request scheduling approaches control the intensity and the order in which concurrent requests from different services should be served, effectively controlling the fraction of resources they consume. In general, the scheduling require some form of prediction of resource consumption for each type of request, either derived analytically [55] [79] or based on observation of past executions [8] [54]. Others, based on control theory [34] [73] [127] [172] use of a closed-loop controller that controls the parameters of the resource provisioning using feedback information from the system.

One limitation of scheduling is that they are generally based on limiting the arrival rate to enforce a certain level of resource consumption and guarantee some service response time. This approach assume either that there is one single resource to manage – generally CPU or bandwidth – or that the allocation for the different resources is proportional. Neither of those assumptions hold in a general case [2].

The partitioning mechanism can be applied at different levels of granularity, depending on the definition of the resources. In [220] servers are assigned to services based on their relative priorities to ensure the QoS objectives per service are met. Cluster reserves [16] extends resource allocation mechanisms provided at the node level to a cluster-wide resource allocation. Virtualization is gaining popularity as a resource partitioning mechanisms as it can be implemented in a non-intrusive way. For example, in VioCluster [179] each service deployed over a virtualized shared cluster composed by virtual machines interconnected by a virtual network.

However, even in the case of consolidated infrastructures based on virtualization – which provide a great level of granularity and control – this isolation is never perfect as some side-effects like cache invalidation [118], network media access collisions and increased disk latency due to head movements can cause some degree of interference.

4.3.5 Membership Management

Request dispatching requires information about the active instances at any given moment. As the number of instances and their location over the (potentially very large) pool of servers may vary over time, traditional membership protocols aimed to support group communication at smaller scales are not well suited. Instead,

[1] In the literature (see for example [101]) this problem is generally studied in the context of service differentiation.

[2] The multiple resources case is different from the multi-layer service case – which can be handled using queuing based schedulers – because the resources are consumed simultaneously and therefore must be co-allocated.

many membership protocols have been proposed that provide each participant with a partial view of the members which is sufficient to support reliable routing in the presence of node failures and churn. Cyclon [203] SCAMP [94] and the peer sampling service [124] use gossip (or epidemic) dissemination of memebership information. RandPeer [151] use a trie data structure to organize the membership information and can cluster peers based on their QoS characteristics, restricting random peer selection within specific QoS clusters to support applications in achieving QoS awareness in neighbor selection.

4.3.6 Request Dispatching

The objective of the request dispatching function in to assign requests to service instances maximizing the possibility that the selected instance can handle it while preserving the QoS requirements. The request dispatching can be applied centralized or decentralized way, using global or local information [42]. It can also be done once per request or in a multi-round (or multi-hop) allocation when a server that cannot handle a request assigned to it, repeats the routing process.

Request dispatching must address two main aspects: how and where redirect the requests and how select the best destination (load balancing).

Request Redirection. There are many alternative mechanism to implement the redirection requests to cluster based web servers [43]. The redirection can be done at different levels of granularity: at the TCP connection, HTTP request, HTTP session (group of related HTTP requests). The redirection can be implemented at different points along the request processing, as shown in figure 4.3:

1. At the web client that originates a request, generally at a reverse proxy acting on behalf of it.
2. At the DNS level, during the address resolution phase, the entity in charge of the request routing is primarily the authoritative DNS server for the web site.
3. At the network level, the client request can be directed by router devices or through multicast/anycast protocols.
4. At the web system level, the entity in charge for the request assignment can be any web server or other dispatching device(s) typically placed in front of the web site architecture.

Depending on the redirection approach, it is possible to implement content-aware – which consider the specific object or service being accessed – or content blind dispatching policies. For example, neither DNS level nor L4 level network redirection allow content-aware because the redirection occurs before the actual HTTP request is sent and therefore the target can't be identified.

Another consideration is that, depending on the placement of the dispatching mechanism, disseminating state information can be more complex. For example, updating DNS entries with accurate load information would require setting very short TTL for the DNS entries to force a frequent refresh, which will impact negatively the performance of the client.

In the case of static content, other considerations apply, mostly to improve cache efficiency, which do not apply to case of dynamic content and are therefore not considered here.

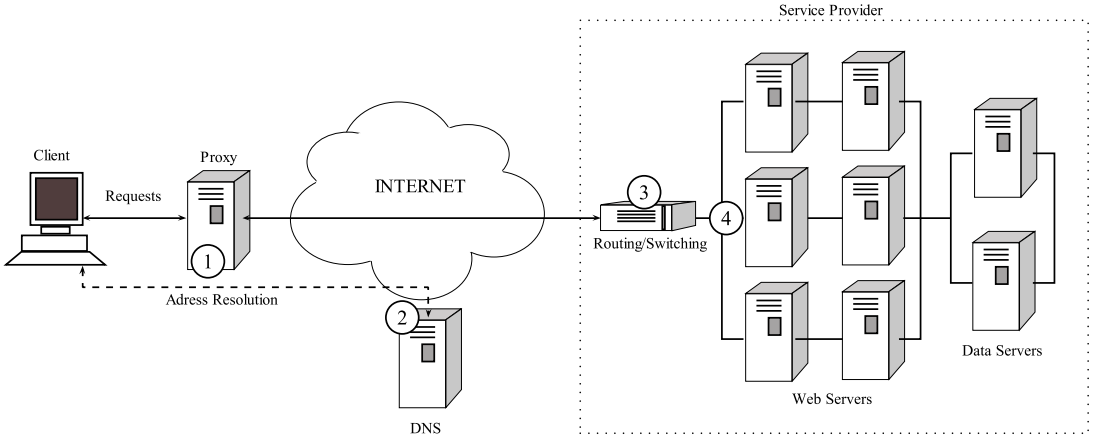


Figure 4.3: Generic architecture for a locally distributed web service.

Load Balancing. Is responsible for distributing requests across web servers, ideally in proportion to their available capacity, to ensure that all servers are used to their full capacity, while no web server is overloaded. It should be noted that load balancing by itself does not guarantee an adequate quality of service on each server, only that each server has a fair amount of work to process.

The problem of request balancing in distributed system has been thoroughly studied in the literature and many heuristics have been proposed using combinations of centralized/decentralized decisions and static that do not require any information about the server's state (for example, a round robin) and dynamic decisions that consider the servers' state in terms of its current load [42] [43] [36]. For dynamic heuristics, one important consideration is the load balancing in the kind of load information being used and the frequency of update of this information. Many load information types have been proposed: the number of active sessions, number of concurrent requests, number of requests in queue, round trip time for requests, total number of bytes transmitted, among others.

4.3.7 Admission Control

Is based on reducing the amount of work the server accepts when it is faced with overload by refusing a fraction of the connections [101]. Simpler admission control approaches refuse all the incoming connections when predefined thresholds are reached [117] or based on performance indicators [57] [34]. In [149] the resource consumption (bandwidth) of clients is controlled by delaying the requests from clients demanding an excessive amount of resources. SEDA [212] implements an finer grained admission control accomplished by internally monitoring the performance of the service, which is decomposed into a set of event-driven stages connected with request queues, to allow the rejection of only of those requests that are limited by the bottleneck components.

4.3.8 Monitoring

To support the management of the service, it is necessary to count with global aggregate information. A number of monitoring mechanisms have been proposed, which adapt to different scenarios. Astrolabe [199] and STAR [119] maintain aggregation trees using an epidemic membership algorithm to locate new nodes and detect failures. Such systems are aimed to scenarios where there exist a sink that gathers all the system information.

Other systems like Willow [199], SOMO [ZSZ03] and DSIMS [215] rely on DHTs to build aggregation trees. One common limitation of those systems is the cost of maintain the DHT; additionally, the attributes being monitored must be predefined (or, even worse, the range value of such attributes must be known).

Epidemic algorithms are also been proposed to maintain global aggregates in very large scale and higly dynamic systems [131] [121]. Their main drawback is the time needed to have an accurate global estimate of the actual values. Epidemics have also been used to maintain a shared bulleting board [12]. Its main drawback is that the average of the information increases rapidly with the size of the cluster and therefore seems not be appropriate for the scenarios of interest.

4.4 Discussion

We have presented a model for adaptive infrastructures for web services, with emphasis on locally distributed deployments. This model proposes a modular separation of the multiple concerns that must be considered when developing such solutions, providing a framework to understand their requirements and clarify their scope.

We have also surveyed solutions from the literature and mapped them to the model, identifying the diverse approaches that have been proposed for each of the concerns we have identified. In this way we facilitate the comparison of alternative approximations for each of the concerns, identifying their advantages and limitations.

Finally, we consider that infrastructures developed following this model will promote the reutilization of solutions and their composition in new ways. Our own proposed infrastructure *e*UDON – presented in chapter 5 – exemplifies its potential.

Chapter 5

*e*UDON an Elastic Utility Driven Overlay Network

The Elastic Utility Driven Overlay Network (*e*UDON) is a middleware for dynamically adapting a service deployed over a highly dynamic large scale infrastructures of non-dedicated servers to ensure it satisfies a target QoS objective. It addresses the following concrete adaptation objectives: a) adapt to fluctuations in the available capacity of each node; b) scale (up and down) the number of instances to match variations in the workload; c) handle the churn of instances as they are activated/deactivated or experiment failures; and d) handle massive failures that require the rapid allocation of multiple instances.

*e*UDON is based on the Collectives framework. Each service deployed over a shared infrastructure behaves as a collective of service instances (the agents) organized using a (composite) overlay. The main function of this collective is the allocation of requests to instances that provide an adequate QoS. This is done by exploiting the modular adaptive routing provided by the Collectives framework to implement the resource location and load balancing heuristics. The main adaptation strategies implemented by *e*UDON are the adaptation of an admission window on each instance to prevent overload, and the elastic assignment of service instances to adapt the global capacity of the system to the fluctuations in the demand. The collectives provides aggregate information used for adaptation strategies.

Following the conceptual architecture for adaptive web services discussed in chapter 4, *e*UDON covers the functions related to Request Allocation: Membership Management, Request Routing, Load Balancing, and Admission Control. The Resource Isolation function is not required for *e*UDON to work and this is one of its main advantages. It also partially covers the function of Service Sizing, as it can dynamically adapt to the workload, but it is limited to do so from a set of already active – but potentially not serving requests– instances. The Monitoring function is covered by the Collectives aggregation function[1].

*e*UDON follows the principles of emergent, utility-driven, and model-less self-adaptation. The adaptation is executed locally on each service instance using simple strategies that really on the current state of the instance and, in a leaser degree, on estimated global aggregate information. The utilization of global information may

[1]This aggregation is not currently implemented. In the experimental evaluation aggregated information is simulated by feeding nodes with the values of the aggregated metric perturbed by a certain random value to simulate the aggregation error.

seem to contradict the emergence principle – which call for local information and coordination. However, this aggregated information can actually be estimated from information obtained from neighbors. It does not need to be either accurate or up to date. The adaptation rules we have chosen are purposefully simple. They are reactive (do dot anticipate changes), probabilistic in nature and depend of few parameters, which basically reflect how aggressively the adaptation process should react to deviations from the desired QoS objective.

5.1 *e*UDON Model

The model of *e*UDON is shown in Figure 5.1. There is a large pool of servers available for diverse services. At any given time, on a subset of those servers there are instances activated to process requests for a service.

Incoming requests for a service are processed through a set of entry-points, which correspond to segments of users with similar QoS requirements. In the context of QoS-based service discovery and composition [150], each entry point represents one of such services with a particular set of QoS attributes. Each entry point is replicated and requests are evenly distributed over the multiple replicas using traditional DNS or level 4 network load balancing techniques [43]. The entry point replicas route the requests to the service instances using the *e*UDON overlay, which handles the load balancing among the instances and the QoS considerations, as explained below.

Each service has a utility function that maps the attributes and execution state of a service instance (e.g. response time, available resources, trustworthiness, reliability, execution cost) to a scalar value that represents the QoS it provides. The QoS offered by an instance may vary over time due to, for example, fluctuations on the load or the available resources of the non-dedicated server it runs on. Each entry point has a QoS requirement defined as the minimum acceptable utility that a service instance must provide to process a service request coming from the entry point.

The adaptation process occurs at three levels whose interplay allow the system to respond to both short and mid term load fluctuations, as well as to failures. First, on the pool of available servers, a certain number of service instances are activated to form the **Service Search Overlay**, which is used to locate instances offering an adequate QoS. The number of active instances is adapted periodically considering the expected workload and the level of redundancy needed to handle short term peaks and eventual failures. Active instances that are not needed are deactivated to save resources and energy. How the number of active instances is calculated and adapted to the workload variations and management policies is not covered in this thesis. Diverse techniques exists as discussed in section 4.3.1.

Having active but not promoted instances can be justified both in cluster and cloud based scenarios. In shared clusters, the active instances which are not promoted for processing requests add little overhead to the cluster. In a cloud scenario, it makes sense to have instances activated for some time even if idle, because of the activation overhead and because computing resources are usually paid by the hour – and therefore a 15 minutes activation cost the same than a full hour activation.

At a second level, from the the active instances, a subset capable of processing the current workload while preserving the expected QoS is promoted to the **Service Routing Overlay**, which has the responsibility of distributing requests to balance the load among the service instances. Those instances which are underutilized or are not meeting the required QoS are demoted, moving back to the Service Search Overlay. This mechanisms

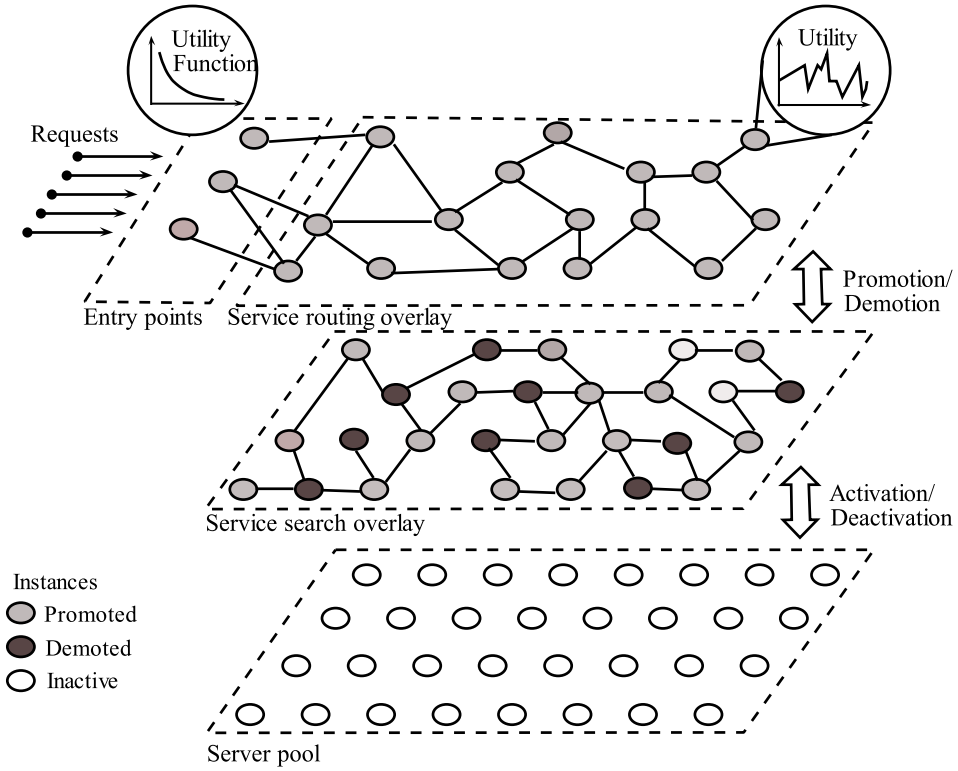


Figure 5.1: Elastic service overlay model.

tries to minimize the number of promoted instances and maximize the resource utilization – an important objective when considering energy efficiency – and reduce the number of hops needed to allocate each request.

Finally, on each instance, an **Adaptive Admission Function** limits the load of the instances to ensure the required QoS, adapting to variations in the server's available resources due to the interference of other services sharing the same physical or virtual machine, or deployed over the same service container.

The routing and search overlays use a push style epidemic algorithm to maintain their topologies, find new (activated, promoted) instances, remove unavailable (failed, demoted, deactivated) instances, and disseminate the current state of instances.

One important consideration in both load balancing and admission control is offering service differentiation for multiple services and service classes. In *e*UDON we address this requirement indirectly. Each service and service class run in a separate instance of *e*UDON. The maintenance of multiple overlays does not necessarily implies increasing the system's overhead, as messages from multiple epidemic overlays may be efficiently delivered by piggybacking them on a single message [201].

The following sections describe the diverse aspects of the model outlined before in more detail and explore the self-adaptation options they offer.

5.2 Utility Function

The utility function is the application dependent component of *e*UDON, as each service may have a different definition on utility depending on its functionality, user expectations, provider's goals, and others. Several utility functions have been proposed in the literature [77] [134]. In our experiments we use a utility function

adapted by the function used in [134] that relates the utility to the deviation of the response time RT from a target maximum response time RT_0:

$$U(RT) = \left[\frac{RT_0 - RT}{RT_0}\right]^{\alpha} \tag{5.1}$$

As shown in Figure 5.2 the coefficient α controls how quickly the utility decreases as the response time approaches the maximum. This function was selected because it can easily be related to metrics obtained from both the analytical and simulation models, making it straightforward to predict and measure the impact of the adaptation decisions in the resulting utility.

However, it is important to stress that the adaptation process considers only the utility function and makes no explicit reference to the underlying response time. This allows us to generalize the results to other utility functions given that they satisfy the basic assumption of being non-increasing with the load of the system, as discussed later in section 5.4.

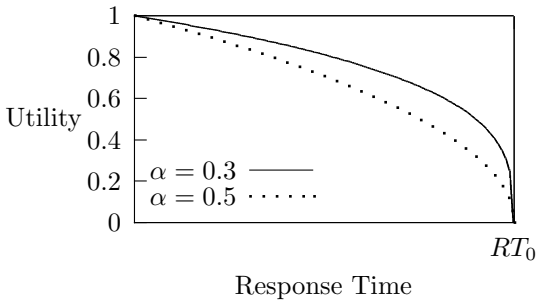


Figure 5.2: Utility Function.

5.3 Overlay Maintenance and Request Routing

*e*UDON uses the generic overlay construction and routing framework mechanisms provided by Collectives, which are shown in Figure 5.3.

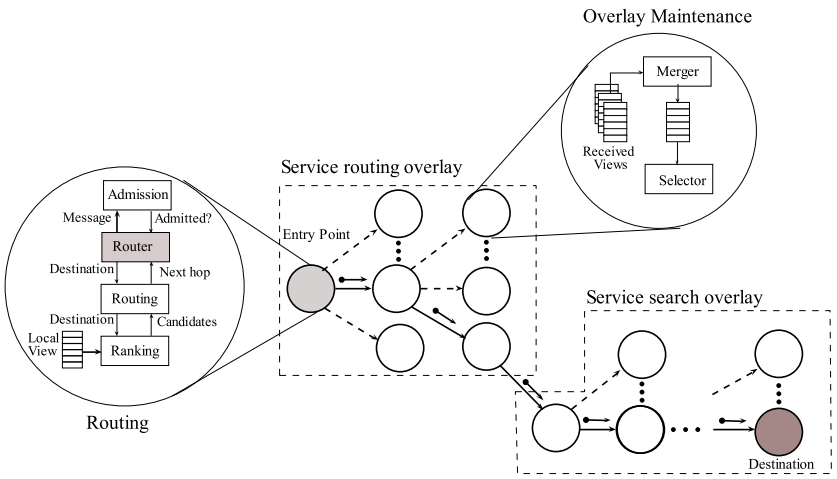


Figure 5.3: Greedy routing over two overlays.

The search and routing overlays are organized using epidemic algorithms. Over each overlay, requests are routed using a generic routing process shown in Figure 5.3. On the reception of a request, the instance uses the *Admission Function* to determine if the node can accept the request for processing. If the request is not accepted, then a *Ranking Function* is used to rank the neighbors and a *Routing Algorithm* selects the next hop in the routing process based on the ranking. This function accommodates the heuristic for selecting the next hop. When a request is routed beyond a predefined number of hops, it is routed using the search overlay, looking for an active (but not promoted) instance capable of serving it.

In *e*UDON the routing and search overlays use different variations of the neighbor selector, ranking heuristic and routing algorithm to meet its particular requirements as explained in sections 5.5 and 5.6. Tables 5.1(a) and 5.1(b) show how alternative versions of these components can be combined according to the unique requirements of each overlay. From the table, it can be appreciated how the proposed modularity offers a great level of adaptability and facilitates the composition of generic building blocks.

Neighbor Selector	Ranking Function	Routing Algorithm
Random	–	Round Robin
Age	–	Round Robin
Age	Capacity	Two Choices
Age	Capacity	Probabilistic

(a) Routing Overlay

Neighbor Selector	Ranking Function	Routing Algorithm
Random	–	Random Walk
Age	Utility	Greedy
Gradient	Utility	Greedy

(b) Search Overlay

Table 5.1: Example of combining Collective's overlay components to create alternative overlays.

5.4 Adaptive Admission Control

*e*UDON uses an adaptive admission function – inspired by those proposed in Quorum [34] and in [20] – that follows the principles of adaptive bounded rationality proposed in [208]. The operation of this function is summarized in Figure 5.4.

The utility of the service instance is periodically monitored and compared with a target QoS objective and the size of the admission window is increased or decreased as needed to correct deviations. The acceptance window is calculated using the following formula:

$$\begin{aligned} \Delta_W &= W_n * (U_n - U_{target}) \\ W &= W_n + \Delta_W \\ W_{n+1} &= \phi * W_n + (1 - \phi) * W \end{aligned} \tag{5.2}$$

The only assumption made by this process is that the utility is non-increasing with respect of the load. That

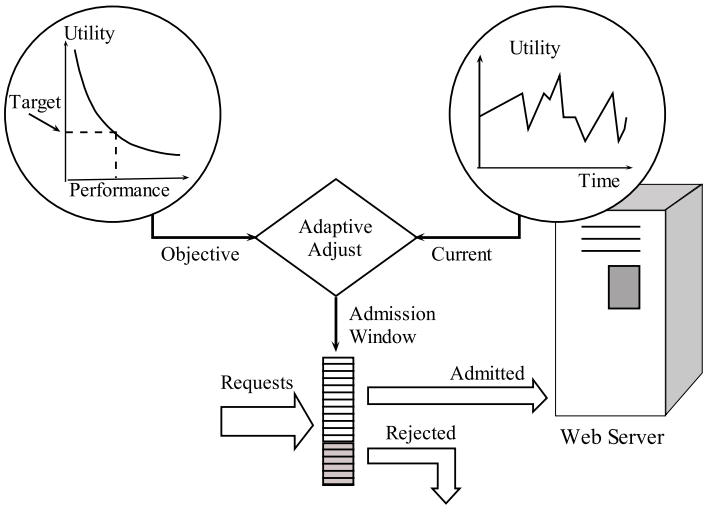


Figure 5.4: Adaptive admission process.

is, that increasing/decreasing the load lowers/rises the utility, given that the rest of the utility related attributes remain equal. One significant advantage of this adaptive admission function is that it does not require any model to estimate the future performance of the service. Moreover, it works even when the resources allocated to a service cannot be reserved and therefore the available capacity fluctuates.

Figure 5.5 shows how as the background load fluctuates so does the capacity of the node to compensate the change in the available CPU and maintains the offered utility close to the target utility U_T. Another interesting result is that the adaptation process not only meets the utility goal, but also achieves a high level of system efficiency, with a total utilization of the node around *0.9*.

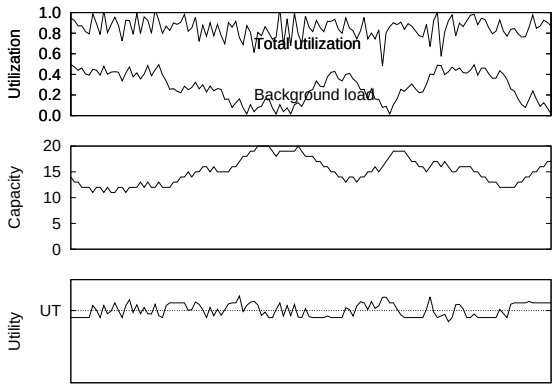


Figure 5.5: An example of the admission self-adaptation process in presence of background workload variations.

5.5 Load Balancing

The main objective of the Routing Overlay is to efficiently deliver requests to a service instance with adequate QoS, while still balancing the request among instances. The load balancing is responsible for distributing requests across instances, ideally in proportion to their available capacity, to ensure that all of them are used to their full capacity, while none is overloaded. It should be noted that load balancing by itself does not guarantee

an adequate quality of service on each instance, only that each one has a fair amount of work to process. The admission control function is responsible for controlling the rate at which instances accept requests to ensure they meet the QoS objectives [41] [36].

In *e*UDON these heuristics are implemented by a combination of a ranking function that orders the neighbors according to a particular criteria, and a routing algorithm that picks one of the neighbors according with their ranking. Two important aspects must be considered in the load balancing in *e*DUON: stale information, and the heterogeneity and variability of servers' capacity.

Stale information is due to the utilization of partial views of the overlay on each instance and epidemic algorithms to probabilistically disseminate information, making impossible to have an accurate global view of the system. The problem of using stale load information was studied in [161] and [64] in the context of a system on which a set of decentralized dispatchers assign requests to a set of servers, using the two random choices heuristic described below. Each dispatcher has full information of the load of the servers, but this information is stale due to delays in the dissemination. Authors found that a random assignment based on stalled load information worked better than either selecting a random server – without considering the load information – or just selecting the server with the lowest load. Also, they discovered that the load received by a server depends only on the server's rank in the list of the dispatcher (based on the load information) and not in the absolute magnitude of the load information nor the difference of this information with respect of other servers. Therefore, stale load information is useful as long as it allows a ranking of servers.

The heterogeneity and variability in capacity of the service instances is due to the fluctuations in the load processed by the non-dedicated node on which they reside. Therefore, knowing the current load for a service instance gives little information about the actual capacity of a instance to receive more requests. In [178] authors proposed to dispatch requests with a probability proportional to the server's maximum capacity, outperforming approaches that use server's load information, with the additional advantages of adapting to the heterogeneity of the servers and also to require updates only when a server's capacity changes.

Under these considerations, we have proposed different alternative heuristics to be evaluated in the context of *UDON*:

- **Round Robin:** Assigns requests orderly among servers. As the list of neighbors change frequently, *e*UDON implements a variation of this approach is selecting the servers randomly with a uniform probability.

- **Random choices[23]:** For each request k servers are randomly selected and a the request is dispatched to the least loaded one, according to some load measure. This heuristic is known to achieve a good load balancing and prevent the herd effect (all requests being assigned to the less loaded servers, actually overloading them), which is likely to occur in decentralized allocations. We consider the two random choices version ($K = 2$) on this heuristic.

- **Probabilistic [20]:** For each request a server is chosen following a probability distribution that is based on the server's state. It can be considered a generalization of the weighted round robin heuristic, amenable to scenarios where the list of server or their probabilities vary over time.

Following the ideas proposed in [178] we have considered therefore as the load information for the Random Choices and Probabilistic heuristics the **Capacity**, defined by the size of the admission window as discussed in section 5.4.

5.6 Resource Discovery

The objective of the Search Overlay is to find with a high probability and a low number of hops, a service instance offering adequate QoS. We evaluated three alternative overlays that use variations of the generic epidemic algorithm described in section 3.2.1 and different ranking functions: the age-based algorithm with a greedy search, a gradient overlay as proposed in [181] and a random overlay as proposed in [5]. In all these overlays, each node selects a random subset of neighbors to disseminate its current state.

In the gradient overlay, each node merges the received information by selecting the nodes that have a utility similar to its own current utility. Requests are routed using a simple greedy routing that exploits the resulting utility gradient. The rationale of this overlay is that, if a node offering a certain QoS cannot handle a request because it has no capacity, it is highly probable that one of its neighbors can handle it offering a similar QoS. This overlay was selected to evaluate if, under the changing conditions of the scenarios of interest, organizing nodes according to their utility offered any advantage.

In the random overlay, each node merges the received information by selecting a random sample of nodes. The routing is done by means of a random walk. This overlay was selected as a baseline to evaluate if the age based overlay offers any systematic improvement over a random search.

5.7 Promotion and Demotion

All active instances participate in the processing of requests. However, to maintain the number of hops needed to process requests low, the routing overlay should minimize the number of instances needed to process the current workload. This is the main function of the promotion/demotion mechanism, as discussed in the next section.

The decision to promote/demote is taken by each instance autonomously based both on its local information, like the rate of request being processed or the server's utilization, and aggregate non-local (potentially global) information like the total workload of the system or the average service rate of other instances.

*e*UDON uses a probabilistic adaptation strategy implemented by two rules based on the service rate (the number of service requests processed per time interval). An instance promotes itself if its service rate is close to the average of the system and demotes itself if it is offering a service rate below the 25th percentile of all instances. These rules where chosen for their simplicity and because they could be easily traceable to the system's status, more than looking for optimality. However, they exhibit a very acceptable behavior as shown in the experimental results. The probabilistic nature of the rules leads to a progressive adaptation preventing that many instances take simultaneously the same decision, over-reacting to a situation and leading to oscillations in the system.

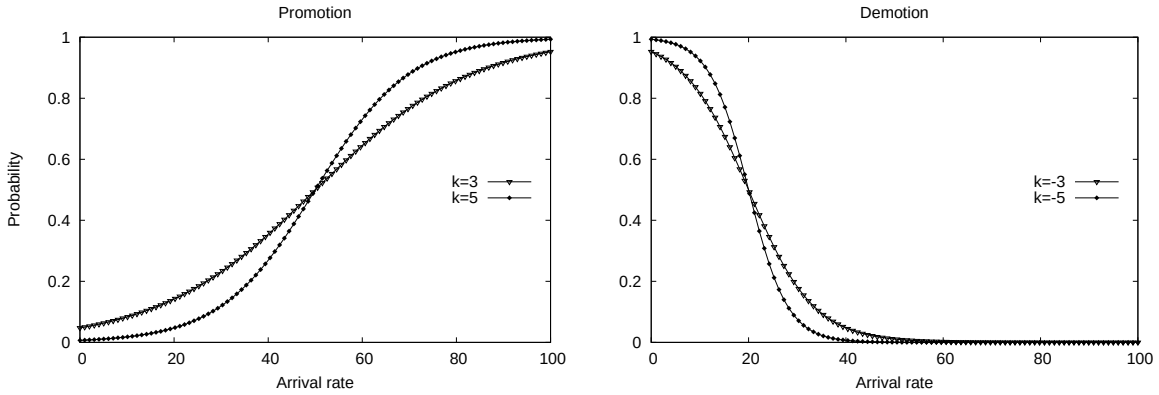
The probability for promotion/demotion an instance is given by the following equation, with a $S - curve$

form:

$$P(S_\epsilon) = \frac{1}{1 + e^{kS_\epsilon}} \tag{5.3}$$

Where $S_\epsilon = (\bar{S} - S)/\bar{S}$ is the deviation of the node's service rate S from the target service rate $\bar{S}$, and k is a parameter that adjust the sensitivity of probability to the service rate. When calculating the probability for demoting, $k > 0$ and for promoting $k < 0$. Figure 5.6 shows this probability for various values of k, and $\bar{S} = 50$ for promotions and $\bar{S} = 20$ for demotions.

An estimation of the global service rate can be obtained by an epidemic aggregation process embedded into the overlay maintenance algorithms [131] [121]. In the simulation described in section 6.1, each instance gets an approximation of these values perturbed by an error factor to simulate the estimation error of the distributed aggregation algorithms.



(a) (b)

Figure 5.6: Promotion/Demotion probability function for diverse values of k.

As the promotion and demotion rules behaves independently of each other, we have added an additional condition to prevent an instance to be continuously promoted/demoted without having the change to stabilize: instances will not run these rules again for the 3 adaptation cycles following a promotion/demotion. This number was empirically obtained by testing multiple options and found to work well in different situations.

5.8 Discussion

One important conclusion we obtained from the design of *e*UDON is that the proposed conceptual architecture for self-adaptive services presented in chapter 4 results useful to effectively decompose the problem in a manageable way and understand the scope of the solution. Moreover, it helps in the selection of alternative approaches and techniques for the diverse aspects involved.

*e*UDON departs from the majority of the work on adaptive web services discussed in that reference model in a variety of ways. In particular, the simplicity of the design and the adaptation rules contrasts with the complexity of other approaches based on models reviewed in chapter 4.

Its model-less approach makes it well suited to form part of a generic middleware stack which could be used to provide self-adaptation to existing services in a non-intrusive way, not requiring a model for the service's

performance. Our approach offers a high level of flexibility by allowing new services to be added and resources reconfigured without requiring any re-engineering of its internal software or hardware infrastructure. Other approaches require models for the workload, the resource utilization or the prediction of the service performance under a certain combination of demand and resource availability.

One additional advantage of the proposed model is that it unifies the responses to different events like service failures and workload surges under a single set of simple adaptation strategies, facilitating the system design.

Additionally, the totally decentralized nature of the solution for load balancing, admission control and request routing, gives room for the scalability in terms of the number of service instances.

Finally, the utilization of both epidemic overlays for organizing the system and disseminating information, in conjunction with bounded rational adaptation rules, makes the system extremely lightweight in terms of both the communication and processing requirements.

As presented in this thesis, *e*UDON has some limitations. Presently we use only instantaneous measurements of an instance's utility for the various adaptation decisions and in particular for the admission acceptance window. As a consequence, even when on average the QoS objective is met, no guarantees of the type "95% of request will have a certain QoS" are possible. However, we consider that we face here the same kind of trade-off that exists in large scale data services like Amazon's Dynamo [69] or Yahoo!'s PNUTS [60] on which the strong consistency guarantees are traded by eventual consistency in exchange of scalability, availability and responsiveness. In the case of *e*UDON the trade-off is between scalability and frugality in one hand, and hard QoS guarantees on the other. Despite this observation, we are exploring the utilization of metrics other than instantaneous measurements to improve the guarantees. For example, the utilization of histograms to estimate the probability of breaking the target QoS.

Anther aspect not considered is the session affinity: the need to map multiple consecutive requests coming from the same source (e.g., same user) to the same service instance, to benefit from session related state caching (for example, maintaining the list of visited items in an online shop). We think that this requirement is becoming less important as web services move towards highly scalable architectures inspired by cloud services, on which this session state is delegated to a distributed caching mechanisms and persistent storage-backed session information [138].

Alternatively, *e*UDON could still be used while maintaining session affinity in the following way: the proposed request routing algorithms can be used when sessions are first established, to find an appropriated service instance. Subsequent request are routed to the selected instance; If the instance can not met the QoS for all the sessions assigned to it an adaptation action could be triggered to redirect a subset of those sessions towards other instances using the same allocation process. In this way, user sessions migrate from one instance to another trying to maintain the expected QoS. One important limitation of this approach is that presently each service is handled by an independent *e*UDON overlay. Therefore, if during a session multiple types of services can be used, the session affinity could be preserved only at the individual service level. One possible solution could be to consider a group of related services as a kind of compound service and handled them in a single *e*UDON overlay. This approach would require a compound utility function to measure the QoS of this service, based on the QoS of the individual services.

Finally, there is the issue of multi-site data centers, a very common setting for large service providers. In

such scenario, the latency in the communication between sites cannot be ignored. Therefore a variation of the basic epidemic protocol for overlay maintenance can be used, selecting the neighbors considering their proximity, but still allowing a margin of randomness to prevent the partitioning of the overlay and allow the routing of request to other sites. Over this overlay the usual request routing algorithms could be used, or it can be also adapted to rank next hops considering the proximity. Both modifications are rather trivial to implement in the framework. The evaluation of this scenario is part of future research.

Chapter 6

Experimental Evaluation

6.1 Experimental Model

In this section we describe the simulation model and the metrics used for the evaluation of the performance of *e*UDON. The intention of this experimental evaluation is to assess the capability of *e*UDON to exhibit an adaptive behavior under different usage scenarios and operational conditions.

The selected scenarios are: A base case with a stable load, which tests the efficiency of the system and its adaptability to fluctuations in the capacity of each node; a peak load scenario on which the system must react to a sudden increase in the load by allocating additional instances; and a failure scenario on which a significant portion of the instances fail simultaneously and must be replaced by other instances.

We implemented a discrete event simulator of the routing and processing of requests to capture very detailed measurements of the system's behavior. In particular, such a simulator was needed to accurately measure the number of hops required to allocate a request, a metric that depends of the concurrence of multiple events: that the request arrives to a instance when the acceptance queue is to its maximum, which in turn depends on the dispatching of the requests previously in the queue. Such detailed modeling is only possible if every significant event is simulated and their timings considered.

One important drawback of modeling to this level of detail is that it limits severely the scale of the experiments. We therefore extensively experimented in medium size configurations and made run test which larger setups to check if the results were consistent. Another approach we considered was using a discrete time simulation on which the "average" behavior of the system during a short period is estimated using analytical model. Even when this model allowed us to scale to up to 32K nodes and some metrics like utilization and allocated demand were accurately estimated, the routing hops had a high error (up to 30% in some test runs) with respect of the discrete event simulator. We therefore decided to trade the scalability of the model in exchange of accuracy.

The simulation considered multiple parameters that define the execution environment. Those parameters are changed to reflect different usage scenarios or operational conditions, while the characteristics of the workload remain mostly invariant. Table 6.1 summarizes the more relevant parameters, which are explained in more detail in subsequent sections of this chapter.

Table 6.1: Simulation parameters.

Parameter	Values	Description
	Overlay	
Servers	**128**, ... 2048	Number of instances
Entries ratio	1:16, **1:32**, 1:48	Ratio of entry points, with respect of the number of instances
Neighbor set	16,**32**,48	Number of neighbors maintained per node in the overlay
Update cycle	**1**,2,3	Frequency of information dissemination (in seconds)
Exchange set	**2**,4,8	Number of neighbors contacted per node on each update cycle
	Utility	
RT	**0.5**	Target Response for request
QoS	**0.7**	Target utility for servers
	Load	
Arrivals	0.75, **100%**, 125%%	Level of system load, with respect of maximum capacity
Variability	**0.10**	Maximum variation of background load per second
Load Maximum	**0.5**, 0.95	Maximum fraction of a server capacity used by background load
	Adaptation	
Adaptation cycle	3	Frequency of the adaptation process (in seconds)
Join probability	.60	Probability of a service instances to join the routing overlay at initialization
K	3 (promotion)	Adaptation probability adjust constant
	3 (demotion)	
$\bar{S}$	60 (promotion)	Target service rate for promotion/demotion
	20 (demotion)	

6.1.1 System modeling

Overlay. We have simulated an idealized network that mimics a large cluster, on which nodes communicate with a uniform delay. The base experimental setup was 128 overlay nodes, with 8 entry points and 120 service instances (a 1:16 ratio). We have conducted test runs with up to 2048 nodes which show analogous results.

Each instance maintains a neighbor set of 32 nodes and contacts 2 of them periodically to exchange information. These values correspond to the optimal trade off between information freshness and communication costs as discussed in experimental results section.

All service instances are initially members of the search overlay, but only a random fraction initially join the routing overlay according to a join probability parameter. The adaptation process dynamically adjust this fraction accordingly to the load.

Service Instances. We model each server instance as a $M/G/1/k*PS$ queuing system [40]. In this model, the server receives requests that follow a Poisson distribution with rate λ. Each request requires a varying service time from the server. The distribution of the service time is unknown (hence the G in the queuing model) with a mean value $\bar{s}$. The server has a single processor and can process up to k concurrent requests. A request will be rejected if the limit of active requests has been reached.

Accepted requests are served using a processor sharing (S) discipline to process them. A request in the queue receives a small quantum of service and is then suspended until every other request has received an identical quantum in a round-robin fashion. When a request has received the amount of service required, it leaves the queue.

This model fits well for web servers like Apache, a well-known and widely used multi-threaded web server, on which each request is handled by its own thread throughout its life cycle. Such servers usually have a maximum number of threads.

It has been shown in [40] that for this model, the expected response time is given by:

$$T = \frac{\rho^{K+1}(K\rho - K - 1) + \rho}{\lambda(1-\rho^K)(1-\rho)} \tag{6.1}$$

where $\rho = \lambda\bar{s}$ is the service rate of the system.

The usage of this model facilitates the comparison of simulation results with analytical estimates. In particular, it allows the estimation of the response time and from it of the utility which explains our selection of the utility function as a function of the response time, as explained in section 5.2.

Arrivals. The service requests arrive following a Poisson distribution and are evenly distributed among the entry points. The arrival rate is calculated using the analytical model for service instances considering the average background load of servers to ensure that the allocation of the workload is feasible, but demands all the available capacity. Therefore, the maximum theoretical allocated demand is of 1.0 and the expected utilization is around 0.95.

QoS. All requests generated in the tests have a maximum response time of 0.5 seconds and the same expected QoS of 0.7 which correspond to a response time of 0.35 seconds approximately, as shown in figure 6.1.

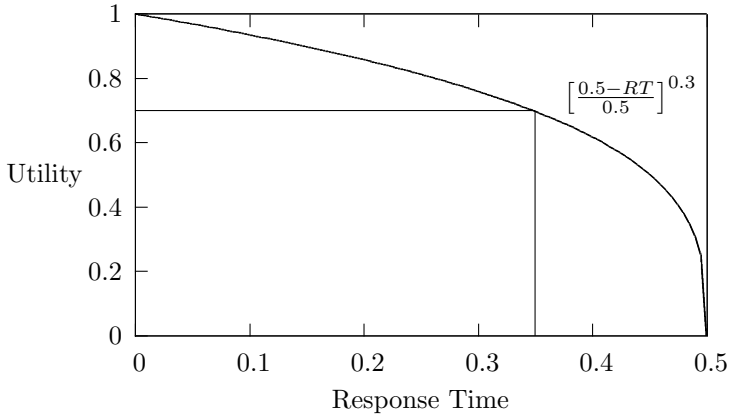
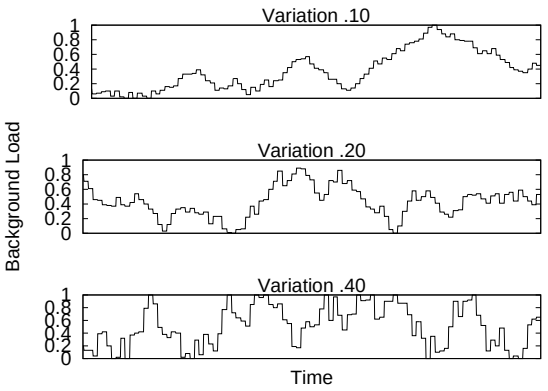
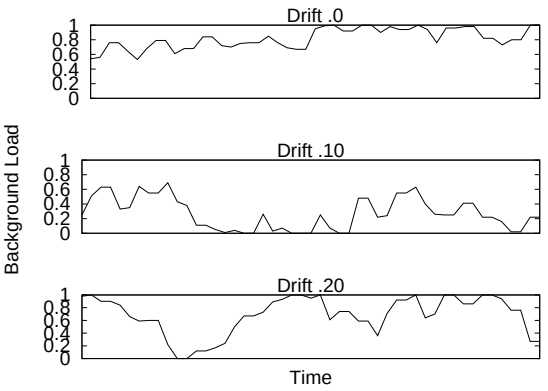


Figure 6.1: Utility Function.

Background Workload. One important aspect in our experiments is the evaluation of the impact of the background load in non-dedicated servers, which impacts the utility that an instance can provide. This load (defined as a fraction of the node's computing power) is modeled as a random variable whose value varies along the time following a random walk with certain *Variability* on each simulation cycle and a *Drift* or tendency to follow a trend upwards or downwards. This model is consistent with the observed behavior of the load on shared servers [169, 216]. Figure 6.2 shows the effect of these parameters in the behavior of one node's utility over time.



(a) Effect of different variations in a node's background load over time. Notice the bigger steps as variation increases.



(b) Effect of different drifts in a node's workload over time, with a fixed variation of ±010. Notice the larger trends as drift increases.

Figure 6.2: Effect of simulation parameters on a node's background load over time.

6.1.2 Metrics

In the evaluation of the proposed adaptation mechanisms we have considered the following metrics, which are related to its main objectives of efficiently delivering requests to an appropriate node to maintain adequate QoS:

Allocated Demand. Measures the fraction of the demand that is actually allocated to a server, before being dropped due to the expiration of its TTL (set to 10 hops in our experiments). This metric measures how effective is the system in allocating requests.

QoS Ratio: Ratio between the target QoS expected by a service request and the actual QoS received. A ratio below 1.0 means that the target was not met, while a ratio over 1.0 means the target was exceeded (which is not necessarily desirable, as it may indicate the server is underutilized).

Utilization: Percentage of the node capacity being used, considering both the background load and the load produced by the service requests. Utilization can be higher than 1.0 if the instance is processing more requests than possible considering the current background load. This metric is relevant as measures how efficiently resources are used.

Routing hops: Number of hops (or retries) needed to allocate a request to a server with adequate utility. It measures how efficient is the mechanism in allocating requests.

In addition, there are other metrics related to the topological characteristics of the overlay worth to be considered:

Age: Measures how fresh is the information at each node. It is calculated at each node by averaging, over all the entries in its neighbor set, how long this information has been circulating on the overlay. Age is measured in terms of update cycles, being a cycle the frequency of the epidemic propagation process in the overlay.

Staleness: Measures how accurate is the information in the neighbor set about other nodes; affects the reliability of the decisions made using this information. The staleness is measured as the average of the (absolute) difference between the value of an attribute in an entry in the neighbor set and the current (real) value of that attribute in the node. It depends on both the Age of the information and the variability of the attribute.

Clustering Coefficient: Measures how tightly nodes reference each other, and therefore, the degree of potential redundant propagation among them. A lower coefficient improves information dissemination, but makes the overlay more susceptible to partitions in the presence of churn.

6.2 Experimental Results

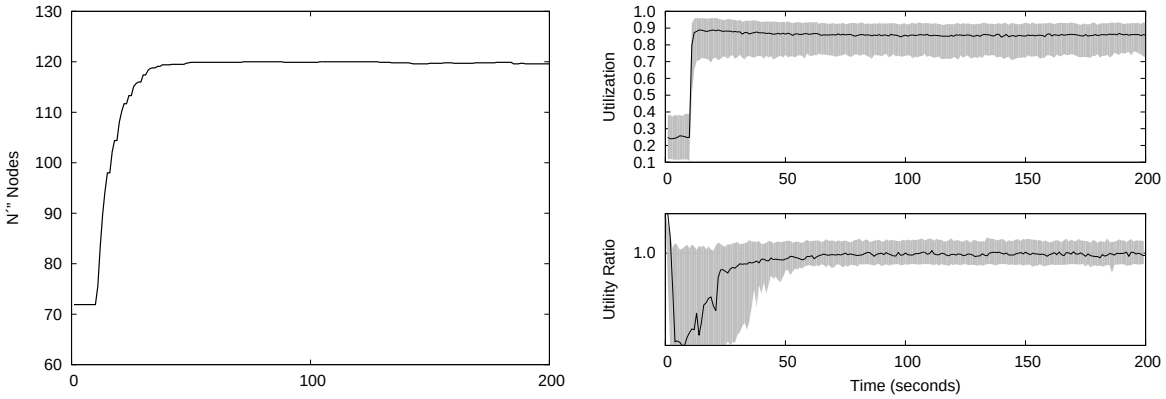
In this section we present the results of the experimental evaluation of key aspects of *e*UDON: Load balancing, service search and elastic adaptation. We also explore how it adapts to different scenarios and its sensitivity to operational conditions and configuration parameters. The results presented correspond, unless the contrary is explicitly indicated, to the average over 10 simulation runs. Each run simulates 200 seconds (300 for the peak load scenario). The ram-up time is of 10 seconds, during which the adaptation process is not active.

In the graphics of experimental results relevant percentiles of the metrics are presented to show their variability.

6.2.1 Base scenario

In the base scenario, the system is submitted to a steady workload that demands all the available capacity to achieve the QoS objective. Initially, a random fraction of instances join the routing overlay ($\approx 60\%$) and the rest are progressively added to satisfy the demand as seen in figure 6.3(a). As shown in figure 6.3(b) it quickly converges to an utility ratio of 1.0, with a small variability. At the same time, the adaptation process also achieves a high level of system efficiency, with a total utilization of system capacity around 0.9.

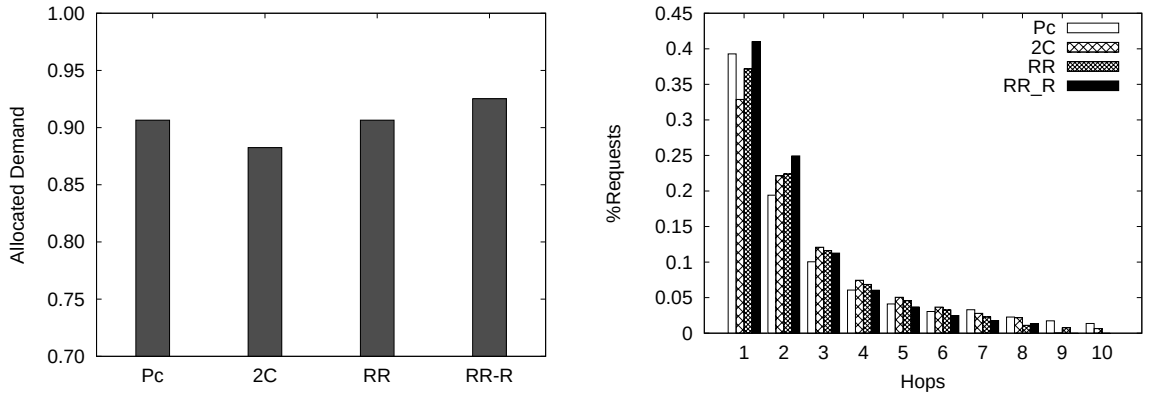
Load Balancing. We studied the efficiency of the balancing heuristics by measuring the fraction of the demand that is actually allocated to a server, before being dropped due to the expiration of its TTL (set to 10 hops in our experiments). The three heuristics: capacity based probabilistic (Pc), two random choices (2C) and round robin (RR) have a very similar fraction of allocated demand (figure 6.4(a)) and similar distribution of hops (figure 6.4(b)). In general, all heuristic were capable of allocating around 90% of the maximum theoretical load with a 75% of requests requiring 3 or less hops.



(a) Evolution of the routing overlay size over time.

(b) Evolution of the Utilization and QoS Ratio (percentiles 5, 50 and 95).

Figure 6.3: Behavior for base scenario.



(a) Allocated demand for different heuristics.

(b) Distribution of routing hops for different heuristics.

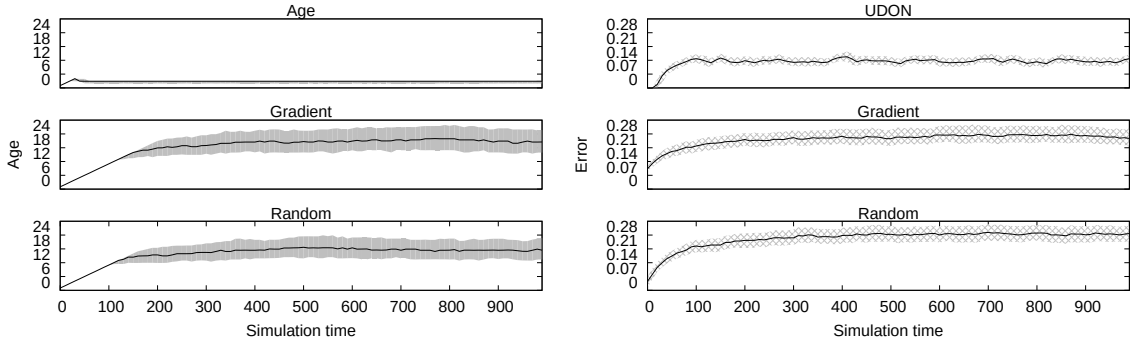
Figure 6.4: Comparison of load balancing heuristics.

From these results, we can conclude that using load information has little effect with respect of not using any load information (round robin), beyond a slightly better change of allocating requests in one hop (shown by Pc). We hypothesize that two aspects account for this result. First, the age based overlay has a clustering coefficient – fraction of neighbors that two neighbor nodes have in common – of around 0.2, which is higher than the coefficient of a random overlay, around 0.1. This higher coefficient affects the capacity of the heuristic of balancing requests among neighbors as the effective fan-out of nodes is lower than the neighbor size. This hypothesis was tested by running the round robin heuristic with a random overlay. The resulting heuristic RR-R achieved a better performance than the round robin with an age based overlay and even improved the Pc heuristic.

In addition, the utilization of admission control in conjunction with the possibility of a multi-hop routing (or, in other words, a multi-round dispatching) minimizes the impact of wrong load balancing decision in the resulting performance. Consider by contrast how in the literature heuristics are generally evaluated in setups where servers will admit the incoming requests up to a fixed maximum and reject the rest, leading to both a high penalty for overloaded servers and a high rate of non-allocated requests. This is compatible with the results found in [42] that the round-robin dispatching when combined with a distributed redirection mechanism

can achieve good and stable performance.

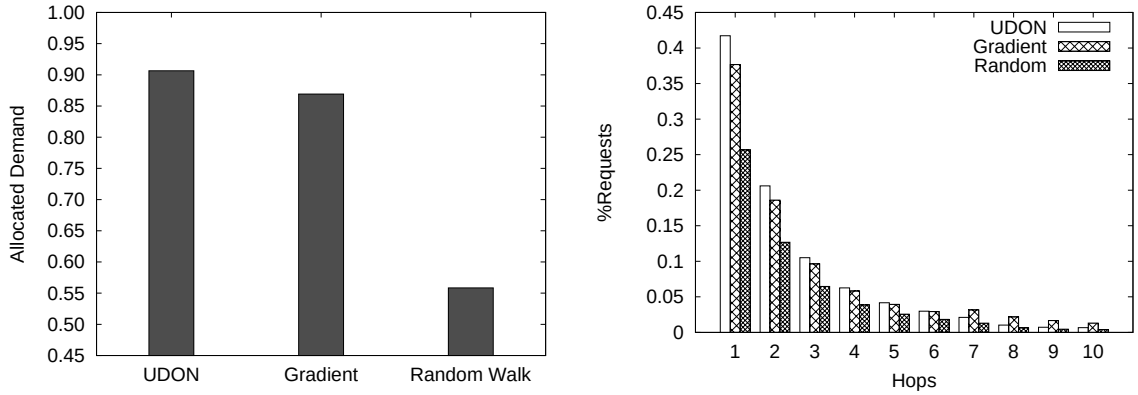
Service Search. We evaluated alternative heuristics for searching instances in the search overlay when the requests cannot be processed in the routing overlay (due to overload or failures). We considered the age based epidemic dissemination with a greedy search (UDON), a gradient epidemic organization with a greedy search and a random epidemic organization with a random walk search. As expected, in the UDON overlay, the Age metric of the information is significantly lower and has less variation than in the other two overlays (figure 6.5(a)), and the information about the utility is much more accurate (figure 6.5(b)). More importantly, those attributes converge very quickly and remains very stable along its execution.



(a) Age (b) Staleness

Figure 6.5: Evolution of metrics for a run with the base experimental setup.

UDON exhibits a better performance than the gradient or random overlays with respect of both the allocated demand and the number of hops (see figure 6.6), allocating requests to a matching instance efficiently even when a high precision (i.e. a low tolerance) is required. This can be explained by the higher accuracy in the information maintained by eUDON (see figure 6.5(b)), which enables it to make better routing decisions.



(a) Allocated Demand (b) Routing Hops

Figure 6.6: Comparison of search heuristics.

The comparable performance of the gradient overlay with respect of the age based overlay can be explained in this scenario by the fact that the admission control function maintains the utility of each node relatively

stable. However, if a different, more variable metric where chosen, like the node's capacity, which varies more significantly (see figure 5.5), then the age overlay maintains its performance while the gradient overlay degrades significantly, as shown in figure 6.7

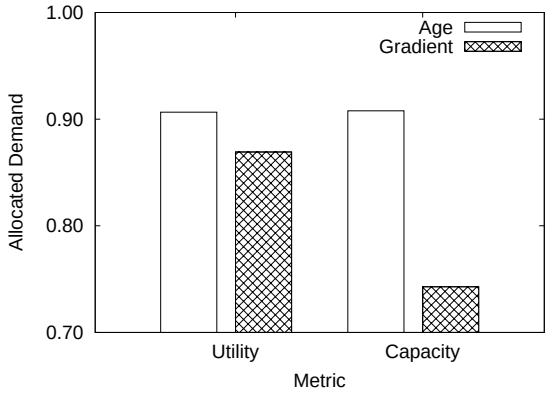
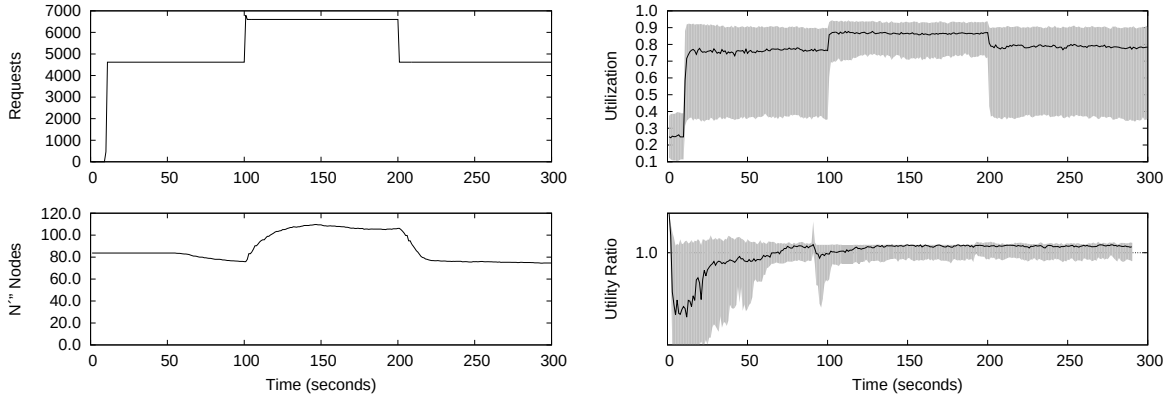


Figure 6.7: Performance of age and gradient overlays when using the node capacity as load information attribute (instead of Utility).

6.2.2 Elastic Adaptation

In this section we describe the different experiments we made to test the adaptability of the system under diverse conditions and usage scenarios.

Peak load scenario. In this scenario, the system is initially submitted to a steady workload that demands 70% of the the available capacity, but at time 100s, an additional load is injected. Figure 6.8 shows how the systems quickly reacts to the surge by promoting more instances. The overall utilization of the system is also increased - the percentile 25 of the Utilization rises significantly – but the QoS Ratio is maintained during this adaptation process. At time 200s, the additional load is removed and the systems returns to the previous state, demoting the instances no longer needed.

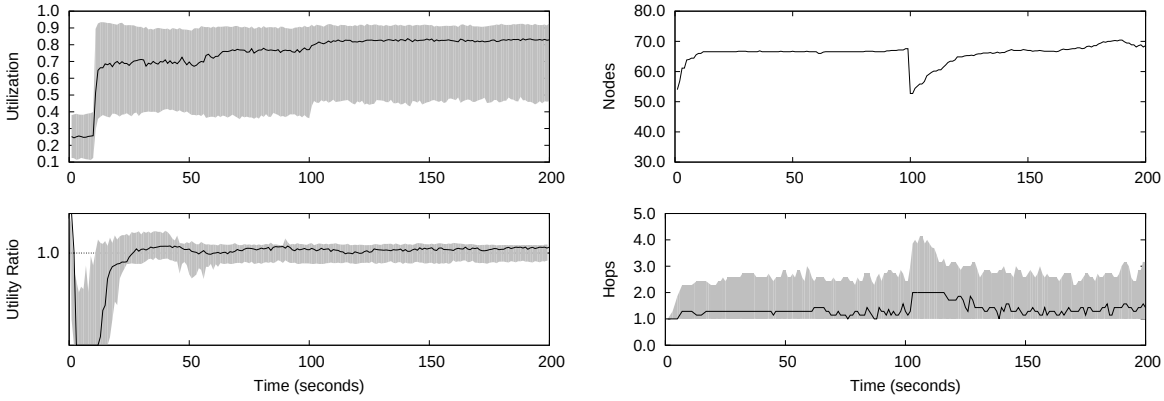


(a) Injected load and number of instances

(b) Utilization and Utility Ratio.

Figure 6.8: Behavior in the peak load scenario.

Failure scenario. In this scenario, the system is submitted to a steady workload that demands a fraction of its capacity. At time 100s, 20% of the promoted instances fail – a correlated failure as expected in clusters. Figure 6.9 shows how the system reacts, incorporating more instances until the system stabilizes. The utility ratio is maintained along this process – except for a short period just after the failure – as requests are routed to nodes in the search overlay; as a consequence, routing hops increase until all the required nodes are promoted to the routing overlay.



(a) Aggregate utilization and utility ratio.

(b) Number of instances and Number of Hops.

Figure 6.9: Evolution of the system behavior in the scenario of a massive failure.

6.2.3 Sensitivity Analysis

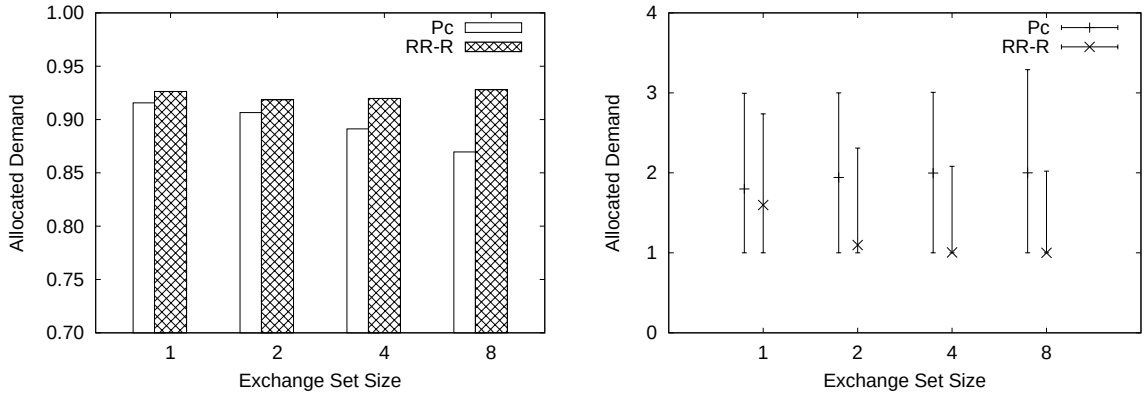
In this section we evaluate the impact of different model parameters and explore the applicability of the proposed approach to different usage conditions.

Scale. We have found that the results are robust with respect of the scale of the system, as we could increase the number of server from 128 to 2048 nodes and observed similar percentage of the significant metrics like allocated demand and number of hops. Moreover, as the instances work exclusively with local information, we expect that the results will hold for larger scale. However, we anticipate that as the scale grows, the number of neighbors maintained for each node (exchange set) must be increased to offer adequate fan-out when balancing requests.

Neighbor and exchange set sizes. These are the two most relevant attributes affecting the epidemic algorithm used to maintain the overlay and disseminate information. With respect of the neighbor set size, in all cases that the ratio between entry points and service instances equals the neighbor set size – that is, the entry points have enough fan-out to cover all the service instances – the system exhibits similar behavior to the base case. When this ratio is significantly lower, the number of hops increases significantly. This is indicative that the neighbor set size is a parameter that can be adapted dynamically by measuring the number of hops needed to allocate requests. This could be particularly important in scenarios with a significantly larger number of instances.

With respect to the size of the exchange set, as seen in figure 6.10, contrary to the intuition, increasing

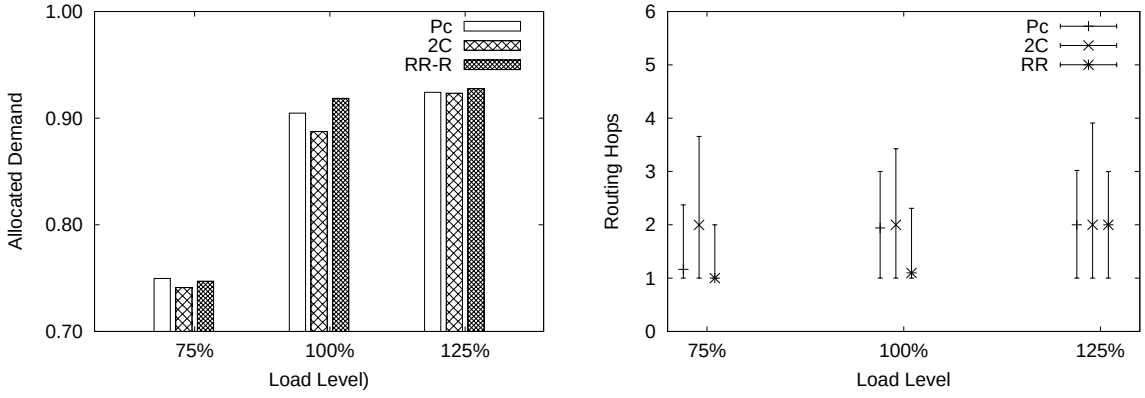
the number of neighbors contacted on each cycle to disseminate the load information has little effect in the performance and, in the case of the capacity based heuristic, can actually degrade it. The reason is that contacting more neighbors on each cycle increases the chance than two nodes be contacted by the same neighbor and therefore their local views to have more elements in common, decreasing the capacity of the system to balance the load. On the contrary, in the Random Round Robin heuristic, the number of hops needed to allocate requests slightly improves as the exchange set size increases. However, this improvement is not justified by the increased communication overhead of contacting more neighbors on each cycle.



(a) Allocated demand (b) Routing hops

Figure 6.10: Effect of exchange set size.

System Load We evaluated the system under loads of 75% and 125% of the maximum theoretical system capacity. When underloaded (load of 75%) the routing hops decreased significantly, with a 75% of requests been allocated with 2 or less hops. When overloaded (load of 125%) the system was capable of maintaining its performance, processing the maximum service rate allowed by its capacity (figure 6.11(a)) while maintaining the target QoS. This is possible due to the admission control function, which adjusts the acceptance of requests to guarantee a target QoS. The main penalty comes from an increased number of hops to allocate requests (figure 6.11(b)). In other words, the system was capable of handling overload without any significant degradation, at the expense of loosing a fraction of load.



(a) Percentage of the maximum theoretical load allocated. (b) Number of hops (percentiles 25, 50 and 75).

Figure 6.11: Sensitivity to load level.

Background load distribution We evaluated the system with different distributions of the background load to simulate systems with different proportions of servers with high load, as seen in figure 6.12. In the base setup this distribution is uniform. We also evaluated the case of a skewed load, with a high proportion of overloaded servers – adjusting the system load to match the total available system capacity. In both scenarios, results are similar to those shown in this report. This result was somehow surprising, as we expected that for skewed case the probabilistic heuristics would outperform more clearly a round robin, as they are basically similar to a weighted round robin, known to fit better in heterogeneous systems.

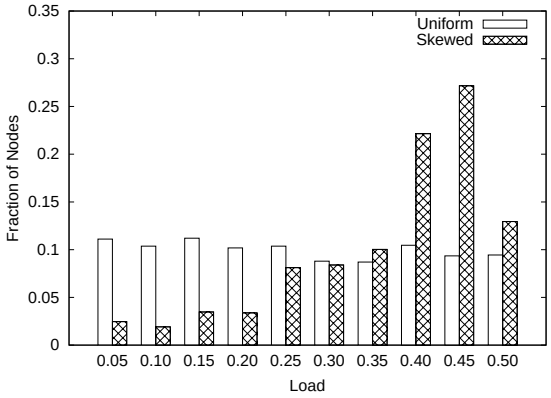


Figure 6.12: Different distributions of the background load in the nodes.

6.2.4 Discussion

The results of the various experiments show that the combination of the epidemic overlay with simple load balancing heuristics and admission control function allow *e*UDON to met the QoS objectives, and achieve a high level of efficiency, and adapt to a diversity of situations.

Moreover, the little differences in the result between different heuristics suggests that these properties depend on the combination of these elements, more than on any of them individually, making the resulting adaption process more robust and its performance more predictable.

None of these results are, when considered in isolation, significantly different from those that can be obtained by other approaches proposed in the literature. What is remarkable is that *e*UDON achieves this performance using a much simpler approach. As discussed in chapter 5, this lower complexity comes from the application of the principles of emergent, utility driven, and model-less self-adaptation.

Chapter 7

Related Work

The development of self-adaptive applications has gained much attention in recent years as a way for tackling increasing complexity. Diverse approaches and concrete implementations have been proposed. In previous chapters we have extensively reviewed alternative approaches regarding self-adaptation in general and adaptive web services in particular. In this chapter we aim to position the proposed solution with respect of other relevant work that share some of our goals. In particular, middleware solutions that aim to provide adaptation capabilities to services. We organize them with respect of the diverse salient aspects of our work to facilitate the comparison.

7.1 Self-adaptation Frameworks

The need for proper abstractions to model and implement self-adaptive systems have been widely recognized in the research community and multiple approaches have been proposed.

In [25] authors propose a model for the separation of concerns when building a self-organizing system. They propose the separation of topology related protocols, which maintain the structure of the system, from functional protocols, which perform application specific functions like aggregate information. However, they do not address the need for a common approach in the implementation of protocols to allow replacing dynamically one protocol without affecting others.

Behavioral skeletons [6] abstract patterns of parallel computations (which are expressed as graphs of components). They encapsulate the assembly logic and the management logic to solve a particular problem and achieve management goals for configuration, optimization, healing and protection. They share some goals with the idea of a Collective and its overlay (abstract out the interaction pattern and provide self-management capabilities) but does not directly support the idea of adapting the structure of the interaction nor the protocols used to support such interaction.

The collective communication among components if also considered in [30] by means of collective communication provider components which encapsulate communication primitives like broadcast. Those components can be dynamically instantiated to specific implementations to adapt to different platforms. It is not clear, however, how such components can achieve run-time adaptation to varying conditions.

In [98] the authors propose the utilization of structural patterns to capture component connectivity topologies and behavioral patterns to capture component interactions which define the temporal and control/data flow dependencies between the components like client/server, master/slave, mobile agent/itinerary. Operators are also provided to compose patterns (structural operators) and to control the execution of a pattern instance (behavioral operators). However, only this high level abstractions are provided and no mapping to particular instantiations are provided. Additionally, patterns are static and cannot adapt to changing conditions during execution.

In [200] authors propose combining a component-based development methodology with structured overlays, using the overlay as a routing mechanisms to facilitate the communication among components. However, the problem of adapting the overlay to the changing conditions that must be faced by the application is not handled. Moreover, the utilization of structured overlays limits the applicability of the approach to cases on which the components can be mapped to an identification space.

7.2 Overlays

The concept of overlay is central to our approach. The development of generic, adaptable overlays has attracted much attention and multiple proposals exist.

In [7] the authors explore the basic constituents of a structured overlay and describe how different overlays can be understood under this model. The proposed model separates the identifier space, the identifier mapping function, the structuring strategy, the routing function and the maintenance strategy.

OverGrid [29] uses a XML based declarative language for the specification of overlay routing protocols and a component-based framework for overlay based applications. iOverlay [148] provides a high performance messaging engine that can be used to implement overlay. Overlayweaver [191] decouples the routing algorithm from the common routing process by designing a programming interface between them, which facilitates the implementation of multiple routing algorithms.

GridKit [99] introduced the concept of pluggable overlay networks and focus on the communications infrastructure required to support multiple interaction types that are demanded by advanced applications (e.g. publish-subscribe, media streaming, peer-to-peer interaction) in a unified, principled and extensible manner.

Closer to our work, GossipKit [153] is an middleware framework that aims to facilitate the development of configurable and reconfigurable middleware supported by multiple (potentially collaborating) gossip protocols that operate in parallel and can adapt to different types of networks. It follows a component-based and event-driven model to integrate the building blocks of the protocols among them and with the application.

The main difference of our work with the frameworks mentioned above is their lack of appropriate abstractions for the application specific adaptation of the overlay construction and the routing protocol, which Collectives incorporates in the form of pluggable functions. Another important difference is that our abstractions and, more importantly, the programming interfaces, are generic enough to be used for both structured and unstructured overlays. This feature offers a significant advantage to developers as the application is isolated from this aspect, which is more related to the efficiency of the implementation than the functionality of the application.

Despite this significant differences, some of the ideas introduced by those frameworks could be integrated into Collectives and form part of future enhancements. In particular GossipKit's event driven interaction among components and iOverlay's efficient messaging mechanisms.

7.3 Request Routing

There have been different efforts to incorporate both epidemic dissemination and the utility functions for routing requests over an overlay. However, they differ significantly in how they apply those concepts and the scenarios on which they are applicable.

In [181] authors propose an utility gradient topology constructed with an epidemic style algorithm on which each node maintains in its neighbor set those peers that have an utility close to its own utility. Over this topology, a simple greedy routing is used to search for the first node with an utility higher than the one required by a request. This overlay seems to be adequate when the utility of the nodes change infrequently but as we show in our experimental results, it is less effective for frequently changing environments.

A middleware for large-scale clusters is proposed in [5] on which nodes self-organize in a per-service overlay using an epidemic algorithm to create a random topology and flood periodically all their neighbors with an load status. Requests are routed randomly to any non overloaded neighbor. If none is found, a random walk is employed to find one. As shown in our experiments, eUDON outperforms the random walk.

7.4 Utility Functions

Utility functions have been extensively used in adaptation and optimization problems. In [209] each application has a utility function that maps a given resource allocation level to the resulting business value; these functions are used by a Global Resource Manager to optimize the allocation of resources to a set of applications.

In [188] and [45] a performance related utility function is used by a request scheduler to order the processing of requests from multiple services classes, so that the resulting aggregate utility is maximized. The main drawback of this approach is its dependency of a cluster level centralized load balancing, making it unpractical to the scales of our systems of interest. Also, it requires the on-line elicitation of the resource consumption profile for each service to adjust the resource allocation, while our approach uses a model-less adaptation.

In [4] an utility function is used by each node to decide which services to instantiate. For a set of applications A the node utility is defined as $\sum_{a\ inA} U_a W_a$, where w_a is the CPU share assigned to the application and u_a is a parameter that weights the applications.

One problem with these utility functions is that are a function of the performance of a proposed resource allocation and therefore relies on a implicit performance model. In our work, we don't assume any prediction model as the utility is calculated over instantaneous measurements of the state of each instance. In that sense, eliciting the utility function for eUDON would be much simpler.

7.5 Load Balancing

The problem of request balancing in distributed system has been thoroughly studied in the literature, initially in the context of job scheduling and more recently in the context of web services [43] [101] [36]. We depart from most of the work in this area in that we use the node's utility as load balancing information instead of the CPU utilization or queue length, as the utility function summarizes multiple parameters into a single value, accommodating heterogeneity and fluctuations in the node's capacity. Additionally, we introduce the utilization of epidemic algorithms as a means to implement randomized allocations.

In [59] authors found that when the request dispatcher does not have full control over all the load that arrives to web servers, policies that use limited load information performed better and were more consistent that those that use no information (round robin) or use detailed load information from the server. In our scenario, we have seen that round robin coupled with a random overlay achieves a better performance. This is possibly due to the adaptive admission control that prevents the overloading of service instances and the multi-hop allocation process that allows re-trying failed allocations.

In [175] three potentially methods for load balancing in large scale Cloud systems are evaluated: a) a nature-inspired algorithm achieving global load balancing via local server actions and a shared adverts board; b) a biased random sampling of the system, setting a node's connectivity accordingly with its capacity; and c) self-organizing the system into clusters of servers with similar capacity that can delegate work to each other. Authors concluded that there appears to be a variation in the best algorithms to use depending on the composition and topology of the server network. One significant limitation of this work is that presented results make impossible to measure the effectiveness or efficiency of the methods with respect to the capacity of the system.

7.6 Admission Control

Quorum [34] uses a model-less adaptive window to control the request rate to enforce QoS guarantees in a multi-class web service. This approach has inspired the self-adaptive admission control used in *e*UDON. The main difference is that Quorum is based on a single metric, response time and is derived from queuing theory models. It is not clear that this approach can be extended to the multi-resource case. *e*UDON uses a utility function that can consider multi-attributes, and its admission control is derived from the principles of bounded rationality. Interestingly, both approaches have converged to a similar solution.

In [20] an admission control is proposed which adaptively determines the request acceptance rate based on the deviation from a target performance and accepts requests with a probability proportional to this fraction. One interesting property of the approach is that the definition of performance is open to the application and multiple metrics can be used simultaneously.

One important difference of *e*UDON with respect of those mechanisms is that they enforce the admission at the system entrance (the load balancer) and require information about all the servers (and the request responses), while *e*UDON applies this policy on each server based only in local information, making it much more scalable.

7.7 Elastic Services

The elasticity in the allocation of resources to web applications has attracted significant attention from different perspectives. One important aspect to consider in the scalability of the solution. The detailed experiments presented in this thesis were conducted with 128 instances, and the same results where confirmed in larger setups of up to 2048 servers. As the instances work exclusively with local information, we expect that the results will hold for even larger scales. To the best of our knowledge, these scales are in general one or two orders of magnitude higher that those reported in most of the literature, which generally show results for, at most, several tens of servers (see for example [20] [33] [129]).

Some notable exceptions to this limitation are presented in [179], [4] and [89] which reach a similar scale but differ from *e*UDON in several key aspects. VioCluster [179] has been evaluated with a simulated cluster of 512 CPUs. It is batch oriented, with and adaptation process running over larger periods and a coarser granularity of tasks than eUDON. The middleware proposed in [4] for self-adaptive web services uses and epidemic overlay and a utility driven placement process. It was evaluated on a simulated cluster of 400 nodes, but the load processed is limited to the 75% of the theoretical capacity of the system, while eUDON achieves over 90% of allocations. In [89] a service placement mechanism modeled after a minority game is proposed, which takes service placement decisions based on local information. The model was simulated with up to 300 nodes. It reaches similar allocated demand than *e*UDON for 100 nodes, but its performance seems to degrade quickly above this size, while eUDON has shown a consistent performance up to 2048 nodes.

Another significant difference is that these systems base the activation and deactivation decisions using a model from the system performance, while *e*UDON is model-less. For example, in [129] the dynamic placement of the instances of multiple applications on a set of server machines is formulated as a two dimensional packaging problem and then solving this problem has a high computational complexity, severely limiting its scalability.

Closer to our work, in [26] nodes are self-organized and sliced according to an application defined metric and the group that represents the "top" slice are selected to form the application's overlay. Its main drawback is that if attributes can change frequently, the slicing must also be continuously updated. However, this process requires the execution of protocols that run over "epochs" of several update cycles in the order of several seconds. This requirement makes this approach unsuitable for the scenarios of interest. In our approach, we integrate this process of updating the set of active nodes into the routing process, making it more responsive to changes. Besides, there's no empirical evidence of the actual performance of the proposed model.

We can conclude that *e*UDON achieves a performance competitive with existing solutions but with a lower complexity and better scalability.

Chapter 8

Conclusions

This thesis addressed the problem of self-adaptation in large scale distributed services. We proposed a set of guiding principles, inspired in economic self-adaptation, and realized them into **Collectives**, a generic framework for self-adaptive applications.

We applied this framework in the development of ***e*UDON**, a middleware for dynamically adapting services deployed on large-scale infrastructures of non-dedicated servers. The resulting system exhibits the intended properties. It can adapt to changing conditions using only local information and local decisions, while maintaining the QoS objectives and achieving an efficient resource utilization. More importantly, the system is highly scalable and resilient to failures, two characteristics that are critical for systems based on commodity hardware clusters. These results prove that the modeling abstractions and the middleware framework provided by Collectives can be effectively used to solve complex self-adaptation problems.

A salient feature of proposed solution is its model-less adaptation approach, which can be applied in scenarios where a model to predict the QoS of a service is not available, or where applying such a model is not feasible due to the dynamism of the environment. In this way, *e*UDON can be used to inject self-adaptation capabilities into a service in a non-intrusive way and without requiring the elicititation of a performance model. This last characteristic is very important as many service-oriented applications are used in scenarios unanticipated at design time, and makes it a candidate to be included in the standard software stack for cloud providers.

The results we have obtained from *e*UDON are very encouraging and open new exciting research opportunities. For example, the fact that *e*UDON has a minimal overhead and converges very quickly to a stable state, makes it a good candidate to provide self-adaptation to the increasingly important category of many-task applications, of which the MapReduce model is a well known example.

The work presented in this thesis has some limitations. Only individual services have been evaluated, while the motivating scenario also considers the composition of basic services. We consider that our work can be easily extended to support service composition following the model proposed in [10]. In this model the QoS of a composite service is defined using an utility function formed with the weighted addition of service quality attributes like response time and availability. This utility function is then decomposed into a series of utility functions which can be evaluated independently for each basic service. This transformation requires the calculation of the minimum and maximum values for each quality attribute for each basic service. These

extreme values can be obtained in UDON by implementing one of several epidemic aggregation algorithms available in the literature. The resulting utility functions can then be used in UDON to drive the selection of service instances.

Finally, as already mentioned, the continuous adaptation of the number of active instances (the activation/deactivation mechanism) is still an open issue we are actively investigating. We envision using a mechanism similar to the one used for promotion/demotion but triggered at the servers to decide which services to activate/deactivate. Taking again inspiration from the economic self-adaptation, we plan to model this process using the same theoretical approach used to model the market entry decision problem. In such problem, a producer must decide which of a set of alternatives markets must enter, based on the expected utility it can obtain considering the demand for each market and the number of potential competitors. Interestingly, there are approximations to this problem based only on a local decisions and global information which could be estimated using epidemic protocols.

We conclude that our work proposes a solution for the requirements of self-adaptation in demanding usage scenarios without introducing additional complexity. In that sense, we believe we make a significant contribution towards the development of future generation service-oriented applications.

Bibliography

[1] S. Abdelwahed, N. Kandasamy, and S. Neema. A control-based framework for self-managing distributed computing systems. In *Proceedings 1st ACM SIGSOFT Workshop on Self-Managed Systems*, NewPort Beach, California USA, November 2004.

[2] T.F. Abdelzaher, J.A. Stackovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3), June 2003.

[3] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Record*, 32(3):29–33, September 2003.

[4] C. Adam and R. Stadler. A middleware design for large-scale clusters offering multiple services. *IEEE electronic Transactions on Network and Service Management (eTNSM)*, 3:1, February 2006.

[5] C. Adam and R. Stadler. Service middleware for self-managing large-scale systems. *IEEE Transactions on Network and Service Management*, 4(3):50–64, 2007.

[6] Marco Aldinucci, Sonia Campa, Marco Danelutto, Marco Vanneschi, Peter Kilpatrick, Patrizio Dazzi, Domenico Laforenza, and Nicola Tonellotto. Behavioural skeletons in gcm: Autonomic management of grid components. In *Proceedings of Euromicro Conference on Parallel, Distributed and Network-Based*, Toulouse, France, 13-15 Feb 2008. IEEE Computer Society.

[7] L.O. Alima, A. Ghodsi, and S. Haridi. A framework for structured peer-to-peer overlay networks. In *Global Computing*, number 3267 in LNCS. Springer, 2005.

[8] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated quality-of-service in web hosting services. In *Workshop on Internet Server Performance (WISP'98)*, 23 June 1998.

[9] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *18th International World Wide Web Conference (WWW2009)*, pages 881–890, 20–24 April 2009.

[10] Mohammad Alrifai, Thomas Risse, Peter Dolog, and Wolfgang Nejdl. A scalable approach for qos-based web service selection (qoscsoa 2008). In *1st International Workshop on Quality-of-Service Concerns in Service Oriented Architectures*, volume 5472 of *Lecture Notes in Computer Science*, pages 190–199. Springer Berlin / Heidelberg, 2008.

[11] Lior Amar, Amnon Barak, Zvi Drezner, and Michael Okun. Randomized gossip algorithms for maintaining a distributed bulletin board with guaranteed age properties. *Concurrency and Computation: Practice and Experience*, 21(15):1907–1927, 2009.

[12] C. Amza, A. Chanda, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic web site benchmarks. In *IEEE International Workshop on Workload Characterization*, pages 3–13, 2002.

[13] Mauro Andreolini and Sara Casolari. Load prediction models in web-based systems. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, 2006.

[14] O. Ardaiz, P. Chacin, I. Chao, F. Freitag, and L. Navarro. An architecture for incorporating decentralized economic models in application layer networks. *Multiagent and Grid Systems*, 1(4):287–295, 2005.

[15] E. Arnautovic, M. Vallé ande, M. Mulvenna, M. Baumgarten, A.M. Hadjiantonis, S.-V. Rehm, M. Müandthel, V. Karyotis, S. Papavassiliou, and K. Stathis. Towards self-managing systems inspired by economic organizations. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 888 –895, oct. 2010.

[16] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. *SIGMETRICS Performance Evaluation Review*, 28:90–101, June 2000.

[17] B.W. Arthur. Complexity and the economy. *Science*, 284(5411):107–109, April 1999.

[18] W. B. Arthur. Inductive reasoning and bounded rationality. *American Economics Review. Papers and Proceedings*, 84:406–411, 1994.

[19] W. Ross Ashby. *Introduction to Cybernetics*. Methuen, 1956.

[20] James Aweya, Michel Ouellette, Delfin Y. Montuno, Bernard Doray, and Kent Felske. An adaptive load balancing scheme for web servers. *International Journal of Network Management*, 12(1):3–39, 2002.

[21] R. Axtell. Effects of interaction topology and activation regime in several multi-agent systems. In *Second International Workshop on Multi-Agent-Based Simulation MABS*, Lecture Notes in Computer Science, pages 33–48. Springer Verlag, 2000.

[22] R. Axtell. Economics as distributed computation. In *Meeting The Challenge Of Social Problems Via Agent-based Simulation: Post-proceedings Of The Second International Workshop On Agent-based Approaches In Economic And Social Complex Systems*, pages 3—-23. Springer, Tokyo, 2003.

[23] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 2000.

[24] O. Babaoglu, G. Canright, A. Deutsch, G. A. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. *ACM Transactions on Autonomic and Adaptive Systems*, 1(1), September 2006.

[25] O. Babaoglu, M. Jelasity, and A. Montresor. Grassroots approach to self-management in large-scale distributed systems. In *Unconventional Programming Paradigms*, volume 3566 of *LNCS*. Springer, 2004.

[26] Ozalp Babaoglu, Márk Jelasity, Anne-Marie Kermarrec, Alberto Montresor, and Maarten van Steen. Managing clouds: a case for a fresh look at large unreliable dynamic networks. *ACM SIGOPS Operating Systems Review*, 40:3, 2006.

[27] Rena Bakhshi, Francois Bonnet, Wan Fokkink, and Boudewijn Haverkort. Formal analysis techniques for gossiping protocols. *ACM SIGOPS Operating Systems Review*, 41(5):28–36, 2007.

[28] Sujoy Basu, Sujata Banerjee, Puneet Sharma, and Sung-Ju Lee. Nodewiz: peer-to-peer resource discovery for grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 9–12 May 2005 2005.

[29] S. Behnel, A. Buchmann, P. Grace, B. Porter, and G. Coulson. A specification-to-deployment architecture for overlay networks. In *Proceedings of the Int. Symposium on Distributed Objects and Applications (DOA)*, Montpellier, France, 2006.

[30] Julien Bigot and Christian Pérez. Enabling collective communications between components. In *Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing*, pages 121–130, New York, NY, USA, 21-22 October 2007. ACM Press.

[31] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao. Able: A toolkit for building multi-agent autonomic systems. *IBM Systems Journal*, 41(3), 2002.

[32] Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, October 2007.

[33] Josep M. Blanquer. *Flexible and Non-Invasive QoS for Scalable Internet Services*. PhD thesis, University of California Santa Barbara., 2005.

[34] Josep M. Blanquer, Antoni Batchelli, Klaus Schauser, and Rich Wolsk. Quorum: Flexible quality of service for internet services. In *2nd Symposium on Networked Systems Design and Implementation (NSDI '05)*, May 2–4 2005.

[35] C. Boutilier, R. Das, J.O. Kephart, G. Tesauro, and W.E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *In Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, August 2003.

[36] H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable web servers. *IEEE Network*, 14(4):58–64, 2000.

[37] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid. In *he Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, 2000.

[38] J. Byers and G. Nasser. Utility-based decision-making in wireless sensor networks. In *Mobile and Ad Hoc Networking and Computing, 2000. MobiHOC. 2000 First Annual Workshop on*, pages 143 –144, 2000.

[39] Kai-Yuan Cai, J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur. An overview of software cybernetics. In *Eleventh Annual International Workshop on Software Technology and Engineering Practice*, 2003.

[40] J. Cao, M. Andersson, C. Nyberg, and M. Kihl. Web server performance modeling using an m/g/1/k*ps queue. In *10th International Conference on Telecommunications*, 2003.

[41] V. Cardellini, M. Colajanni, and P.S Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, May 1999.

[42] V. Cardellini, M. Colajanni, and P.S. Yu. Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):355–368, April 2003.

[43] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.

[44] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an economy-based optimisation of file access and replication on a data grid. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID*, page 340, may 2002.

[45] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade. Utility-based placement of dynamic web applications with fairness goals. In *IEEE Network Operations and Management Symposium, 2008. NOMS 2008*, 2008.

[46] P. Chacin and L. Navarro. Collectives: A framework for self-adaptive p2p application. In *Proceedings of the 6th Workshop on Adaptive and Reflexive Middleware (ARM2007)*, New Port Beach, California, USA., November 26 2007.

[47] Pablo Chacin, Felix Freitag, Leandro Navarro, Isaac Chao, and Oscar Ardaiz. Integration of decentralized economic models for resource self-management in application layer networks. In Ioannis Stavrakakis and Michael Smirnov, editors, *Autonomic Communication*, volume 3854 of *Lecture Notes in Computer Science*, pages 214–225. Springer Berlin / Heidelberg, 2006.

[48] Pablo Chacin, Liviu Joita, Björn Schnizler, and Felix Freitag. Flexible architecture for supporting auctions in grids. In *Workshop in Smart Grid Technologies on the Internacional Conference on Autonomic Computing (ICAC 2006)*, 2006.

[49] Pablo Chacin, Xavier Leon, Rene Brunner, Felix Freitag, and Leandro Navarro. Core services for grid markets. In Thierry Priol and Marco Vanneschi, editors, *From Grids to Service and Pervasive Computing*, pages 205–215. Springer US, 2008.

[50] Pablo Chacin and Leando Navarro. Utility driven elastic services. In *Proceedings 11th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 6723 of *Lecture Notes in Computer Science*. Jun 6-9 2011.

[51] Pablo Chacin, Leando Navarro, and Pedro Garcia Lopez. Utility driven service routing over large scale infrastructures. In *Towards a Service-Based Internet. Proceedings of the Thirds European Conference ServiceWave*, volume 6481. Springer Berlin / Heidelberg, 2010.

[52] Pablo Chacin, Leando Navarro, and Pedro Garcia Lopez. Load balancing on large-scale service infrastructures. Technical Report UPC-DAC-RR-XCSD-2011-1, Polytechnic University of Catalonia, Computer Architecture Deparment. Computer Networks and Distributed Systen Group., 2011.

[53] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the benefits of resource multiplexing in on-demand data centers. In *First ACM Workshop on Algorithms and Architectures for Self-Managing Systems*, 2003.

[54] A. Chandra, P. Pradhan, R. Tewari, S. Sahu, and P. Shenoy. An observation-based approach towards self-managing web servers. *Computer Communications*, 29(8):1174–1188, 2006.

[55] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Quality of Service — IWQoS*, volume 2707, pages 381–398. Springer Berlin / Heidelberg, 2003.

[56] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A case study in building layered dht applications. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 97–108, 2005.

[57] L. Cherkasova and P. Phaal. Session-based admission control: a mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, 51(6):669–685, jun 2002.

[58] Francis C. Chu and Joseph Y. Halpern. Great expectations. part ii: generalized expected utility as a universal decision rule. *Artificial Intelligence*, 159(1-2):207–229, November 2004.

[59] M. Colajanni, P.S. Yu, and D.M. Dias. Scheduling algorithms for distributed web servers. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, 169–176, 1997.

[60] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1:1277–1288, August 2008.

[61] P. Costa, V. Gramoli, M. Jelasity, G.P. Jesi, E. Le Merrer, A. Montresor, and L. Querzoni. Exploring the interdisciplinary connections of gossip-based systems. *ACM SIGOPS Operating Systems Review*, 41(5):51–60, 2007.

[62] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 181–194, 2001.

[63] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*. 2003.

[64] M. Dahlin. Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1033–1047, 2000.

[65] D. De Roure. Future for european grids: Grids and service oriented knowledge utilities vision and research directions 2010 and beyond. Technical report, European Comission, 2006.

[66] T. De Wolf and T. Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. In S. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, editors, *Engineering Self Organising Systems: Methodologies and Applications*, volume 3464 of *Lecture Notes in Computer Science*. May 2005.

[67] T. De Wolf and T. Holvoet. *Autonomic Computing: Concepts, Infrastructure, and Applications*, chapter A Taxonomy for Self-* Properties in Decentralised Autonomic Computing, pages 101–120. CRC Press, 2007.

[68] Tom De Wolf and Tom Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In *Engineering Self-Organising Systems*, volume 4335 of *Lecture Notes in Computer Science*, pages 28–49. Springer Berlin / Heidelberg, 2007.

[69] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, October 2007.

[70] Paul deGrandis and Giuseppe Valetto. Elicitation and utilization of application-level utility functions. In *Proceedings of the 6th international conference on Autonomic computing*, 2009.

[71] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM.

[72] Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G.E. Kaiser, and D. Phung. A control theory foundation for self-managing computing systems. *IEEE Journal on Selected Areas in Communications*, 23(12):2213–2222, December 2005.

[73] Yixin Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury. Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. In *Proceedings IEEE/IFIP Network Operations and Management Symposium NOMS 2002*, pages 219 – 234, 2002.

[74] Peter A. Dinda and David R. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, December 2000.

[75] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao. Autonomia: an autonomic computing environment. In *Proceedings of the 2003 IEEE International Conference on Performance, Computing, and Communications*. IEEE International, 2003.

[76] J. Dowling and V. Cahill. Self-managed decentralised systems using k-components and collaborative reinforcement learning. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. ACM Press, 2004.

[77] J. Dowling, J. Sacha, and S. Haridi. Improving ice service selection in a p2p system using the gradient topology. In *First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 285–288, July 2007.

[78] Jon Doyle. Rationality and its role in reasoning. *Computational Intelligence*, 8(2):376–409, May 1992.

[79] Ron Doyle, Jeff Chase, Omer Asad, Wei Jin, and Amin Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[80] James S. Dyer. Maut — multiattribute utility theory. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, volume 78 of *International Series in Operations Research & Management Science*. Springer New York, 2006.

[81] Bruce Edmonds. *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, chapter Using the Experimental Method to Produce Reliable Self-Organised Systems. Springer, 2005.

[82] A.E. Eiben. Evolutionary computing and autonomic computing: Shared problems, shared solutions? In *Self-star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*. Springer, Berlin / Heidelberg, 2005.

[83] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, November 2003.

[84] Patrick Eugster, Pascal Felber, and Fabrice Le Fessant. The "art" of programming gossip-based systems. *SIGOPS Oper. Syst. Rev.*, 41:37–42, October 2007.

[85] P.T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, May 2004.

[86] T. Eymann, B. Padovan, and D. Schoder. The catallaxy as a new paradigm for the design of information systems. In *Proceedings of the 16th IFIP World Computer Congress, Conference on Intelligent Information Processing*, Beijing, China, 2000.

[87] T. Eymann, M. Reinicke, F. Freitag, L. Navarro, O. Ardaiz, and P. Artigas. A hayekian self-organization approach to service allocation in computing systems. *Advanced Engineering Informatics*, 19(3):223–233, 2005.

[88] Giorgio Fagiolo. Endogenous neighborhood formation in a local coordination model with negative network externalities. *Journal of Economic Dynamics and Control*, 29(1-2):297–319, Jan 2005.

[89] Jeroen Famaey, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. Dynamic overlay node activation algorithms for large-scale service deployments. In *Managing Large-Scale Service Deployment*, volume 5273 of *Lecture Notes in Computer Science*, pages 14–27. Springer Berlin / Heidelberg, 2008.

[90] D.F. Ferguson. *The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms in Distributed Systems*. PhD thesis, Columbia University, 1989.

[91] D.F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. *Market-Based Control: A Paradigm for Distributed Resource Allocation.*, chapter Economic models for allocating resources in computer systems. World Scientific Press, 1996.

[92] I. Foster, C Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(2), 2002.

[93] Wojciech Galuba and Karl Aberer. Generic emergent overlays in arbitrary peer identifier spaces. In *Self-Organizing Systems*, volume 4725 of *LNCS*. Springer, 2007.

[94] Ayalvadi Ganesh, Anne-Marie Kermarrec, and Laurent MassouliÃ©. Scamp- peer-to-peer lightweight membership service for large-scale group communication. In Jon Crowcroft and Markus Hofmann, editors, *Networked Group Communication*, volume 2233 of *Lecture Notes in Computer Science*, pages 44–55. Springer Berlin / Heidelberg, 2001.

[95] D. Garlan and D. E. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4), April 1995.

[96] K. Geihs. Middleware challenges ahead. *Computer*, 34(6), 2001.

[97] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In D. Garlan, J. Kramer, , and A. Wolf, editors, *Proceedings of the First Workshop on Self-Healing Systems (WOSS)*. ACM Press, November 18-19 2002.

[98] M. Cecilia Gomes, Jose C. Cunha, and Omer F. Rana. Pattern operators for grid environments. *Scientific Programming Journal*, 11(3):237 – 261, 2003.

[99] Paul Grace, Geoff Coulson, Gordon Blair, Laurent Mathy, Wai Yeung, Wei Cai, David Duce, and Chris Cooper. Gridkit: Pluggable overlay networks for grid computing. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3291 of *Lecture Notes in Computer Science*, pages 1463–1481. Springer Berlin / Heidelberg, 2004.

[100] J. Gradwell and J. Padget. Distributed combinatorial resource scheduling. In *Proceedings AAMAS Workshop on Smart Grid Technologies (SGT-2005)*, 2005.

[101] Jordi Guitart, Jordi Torres, and Eduard Ayguadé. A survey on performance management for internet applications. *Concurrency and Computation: Practice and Experience*, 22(1):69–106, 2009.

[102] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 160–169. Springer Berlin / Heidelberg, 2003.

[103] D. Hales and S. Arteconi. Slacer: a self-organizing protocol for coordination in peer-to-peer networks. *Intelligent Systems, IEEE*, 21(2):29 – 35, march-april 2006.

[104] D. Hales and B. Edmonds. Applying a socially inspired technique (tags) to improve cooperation in p2p networks. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(3):385 – 395, may 2005.

[105] Maya Haridasan and Robbert van Renesse. Gossip-based distribution estimation in peer-to-peer networks. In *The 7th International Workshop on Peer-to-Peer Systems*, February 25–26 2008.

[106] S. Hassas, G. Di Marzo Serugendo, A. Karageorgos, and C. Castelfranchi. Self-organising mechanisms from social and business/economics approaches. *Informatica*, 30:63–71, 2006.

[107] M. Hauswirth and R Schmidt. An overlay network for resource discovery in grids. In *Proceedings. Sixteenth International Workshop on Database and Expert Systems Applications*, pages 343–348, 26–26 Aug 2005.

[108] F.A. Hayek. Competition as a discovery procedure. *The quarterly journal of austrian economics*, 5(3):9–23, 2002.

[109] J.L. Hellerstein, F. Zhang, and P. Shahabuddin. Characterizing normal operation of a web server: Application to workload forecasting and problem detection. In *Proceedings of the Computer Measurement Group*, 98.

[110] S. Herrmann. Object teams: Improving modularity for crosscutting collaborations. In *Architectures, Services, and Applications for a Networked World: International Conference NetOb-jectDays*, volume 2592 of *LNCS*. 2002.

[111] Steven Horwitz. Catallaxy, competition, and 21st century capitalism: An agenda for economics. In *First Conference on The Future of Heterodox Economics (Annual Conference of Confederation for the Advancement of Pluralism in Economics)*, 2003.

[112] Hung-Chang Hsiao, Mark Baker, and Chung-Ta King. A peer-to-peer mechanism for resource location and allocation over the grid. In *Parallel and Distributed Processing and Applications*, volume 3358 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005.

[113] B. A. Huberman and T. Hogg. Distributed computation as an economic system. *Journal of Economic Perspectives*, 9(1):141–152, 1995.

[114] L. Hurwicz. The design of mechanisms for resource allocation. *The American Economic Review*, 63:1–30, 1973.

[115] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid en-vironments. In *Proceedings of the Second international Workshop on Grid Computing*, volume 2242 of *LNCS*, pages 51–62. Springer, 2001.

[116] Arun K. Iyengar, Mark S. Squillante, and Li Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2:85–100, 1999.

[117] R. Iyer, V. Tewari, and K. Kant. Overload control mechanisms for web servers. In *Workshop on Performance and QoS of Next Generation Networks*, 2000.

[118] Ravi Iyer, Ramesh Illikkal, Omesh Tickoo, Li Zhao, Padma Apparao, and Don Newell. Vm3: Measuring, modeling and managing vm shared resources. *Comput. Netw.*, 53:2873–2887, December 2009.

[119] Navendu Jain, Dmitry Kit, Prince Mahajan, Praveen Yalagandula, Mike Dahlin, and Yin Zhang. Star: self-tuning aggregation for scalable monitoring. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 962–973. VLDB Endowment, 2007.

[120] Michal Jakob. *Multi-Agent Service Selecction Competitive Resource-Constrained Environments*. Phd thesis, Czech Technical University in Prague. Faculty of Electrical Engineering. Department of Cybernetics, 2008.

[121] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–259, 2005.

[122] Márk Jelasity and Ozalp Babaoglu. T-man: Gossip-based overlay topology management. In *Engineering Self-Organising Systems*, volume 3910. 2006.

[123] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 79–98, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[124] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):8, August 2007.

[125] N.R. Jennings. Building complex distributed systems: The case for an agent based approach. *Communications of the ACM*, 44(4), 2001.

[126] K. Kar, S. Sarkar, and L. Tassiulas. A simple rate control algorithm for max total user utility. In *Proceedings of the IEEE 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 133 –141 vol.1, 2001.

[127] M. Karlsson, Xiaoyun Zhu, and C. Karamanolis. An adaptive optimal controller for non-intrusive performance differentiation in computing services. In *Control and Automation, 2005. ICCA '05. International Conference on*, volume 2, pages 709 –714 Vol. 2, june 2005.

[128] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 565 –574, 2000.

[129] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *Proceedings of the 15th international conference on World Wide Web*, pages 595–604. ACM, 2006.

[130] Terence Kelly. Utility-directed allocation. Technical Report HPL-2003-115, Internet Systems and Storage Laboratory. HP Laboratories Palo Alto, June 2003.

[131] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, Octuber 11–14 2003.

[132] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kasselman, R. Aydt, D. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski. Toward a framework for preparing and executing adaptive grid programs. In *Proceedings of NSF Next Generation Systems Program Workshop*, Fort Lauderdale, FL, USA, 2002.

[133] J. O. Kephart. Research challenges of autonomic computing. In *Proceedings of the 27th international Conference on Software Engineering*, St. Louis, MO, USA, May 15-21 2005.

[134] Jeffrey O. Kephart and Rajarshi Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, January 2007.

[135] J.O. Kephart and M.D. Chess. The vision of autonomic computing. *Computer*, 31(1):41–50, 2003.

[136] Alan Kirman. The economy as an evolving network. *Journal of Evolutionary Economics*, 7(4):339–353, 1997.

[137] Israel M. Kirzner. Coordination as a criterion for economic "goodness". *Constitutional Political Economy*, 9(4):289–301, 1998.

[138] Donald Kossmann, Tim Kraska, and Simon Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *oceedings of the 2010 international conference on Management of data SIGMOD'10*, pages 579–590, 2010.

[139] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *International Conference on Software Engineering*, Washington, DC, USA, 2007. IEEE Computer Society.

[140] J.F. Kurose and R. Sima. A microeconomic approach for optimal resource allocation in distributed computer systems. *IEEE Transactions on Computer*, 38(5):705–717, 1989.

[141] M. Kwon and S. Fahmy. Synergy: an overlay internetworking architecture. In *Proceedings. 14th International Conference on Computer Communications and Networks (ICCCN).*, pages 401 – 406, oct. 2005.

[142] Robert Laddaga. Active software. In *First International Workshop on Self-Adaptive Software (IWSAS 2000). Revised Papers*, volume 1936 of *Lecture Notes in Computer Science*. Springer, Oxford, UK, April 2000.

[143] K. Lai, B. A. Huberman, and L. Fine. Tycoon: A distributed market-based resource allocation system. Technical Report cs.DC/0404013, HP Lab, Palo Alto, Apr. 2004.

[144] Kevin Lai. Markets are dead, long live markets. *ACM SIGecom Exchanges*, 5(4):1–10, July 2005.

[145] A.G. Laws, A. Taleb-Bendiab, S.J. Wade, and D. Reilly. From wetware to software: A cybernetic perspective of self-adaptive software. volume 2614 of *Lecture Notes in Computer Science*, pages 257–280. Springer, january 2003.

[146] Kevin Lee, Norman W. Paton, Rizos Sakellariou, and Alvaro A. A. Fernandes. Utility driven adaptive workflow execution. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[147] Xavier León, Tuan Anh Trinh, and Leandro Navarro. Using economic regulation to prevent resource congestion in large-scale shared infrastructures. *Future Gener. Comput. Syst.*, 26:599–607, April 2010.

[148] B. Li, J. Guo, and M. Wang. ioverlay: a lightweight middleware infrastructure for overlay application implementations. In *Proceedings of the 5th ACM/IFIP/USENIX international Conference on Middleware*, Toronto, Canada, October 18 - 22 2004.

[149] K. Li and S. Jamin. A measurement-based admission-controlled web server. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 651–659, 2000.

[150] Qing Li, An Liu, Hai Liu, Baoping Lin, Liusheng Huang, and Naijie Gu. Web services provision: solutions, challenges and opportunities (invited paper). In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, pages 80–87, New York, NY, USA, 2009. ACM.

[151] J. Liang and K. Nahrstedt. Randpeer: Membership management for qos sensitive peer-to-peer applications. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, april 2006.

[152] P. Lin, A. MacArthur, and J. Leaney. Defining autonomic computing: A software engineering perspective. In *Proceedings of the 2005 Australian conference on Software Engineering*, pages 88–97. IEEE Computer Society., 2005.

[153] Shen Lin, François Taïani, and Gordon Blair. Facilitating gossip programming with the gossipkit framework. In *Distributed Applications and Interoperable Systems*. Springer Berlin / Heidelberg, 2008.

[154] Shen Lin, François Taïani, and Gordon Blair. Exploiting synergies between coexisting overlays. In *Distributed Applications and Interoperable Systems*, volume 5523 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2009.

[155] Huan Liu and Sewook Wee. Web server farm in the cloud: Performance evaluation and dynamic architecture. In *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 2009.

[156] A. Lomi, E.R Larsen, and A. Ginsberg. Adaptive learning in organizations: A system dynamics-based exploration. *Journal of Management*, 23(4):561–582, 1997.

[157] Ka man Lam and Ho fung Leung. An adaptive strategy for resource allocation modeled as minority game. In *First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 193–204, 9–11 Jul 2007.

[158] Hermann De Meer and Christian Koppen. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter Characterization of Self-Organization, pages 227–246. Springer, 2005.

[159] D. A. Menasce. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[160] Alex C. Meng. On evaluating self-adaptive software. In *Proceedings of the first international workshop on Self-adaptive software*, pages 65–74, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[161] Michael Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1):6–20, January 2000.

[162] J. C. Mogul. Emergent (mis)behavior vs. complex software systems. In *Proceedings of the 2006 Eurosys Conference*, Leuven, Belgium, April 18-21 2006.

[163] A. Montresor and R. Zandonati. Absolute slicing in peer-to-peer systems. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*, pages 1 –8, april 2008.

[164] Tracy Mullen and Michael P. Wellman. Some issues in the design of market-oriented agents. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II, Agent Theories, Proceedings of Architectures, and Languages Workshop (, IJCAI '95)*, volume 1037 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 1996.

[165] R. A Nagpal. Catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering*, volume 2977 of *Lecture Notes in Computer Science*. Lecture Notes in Computer Science, 2004.

[166] J. Nakai and Rob F. Van Der Wijngaart. Applicability of markets to global scheduling in grids. NAS Tech Report NAS-03-004, ASA Ames Research Center, 2003.

[167] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications*, August 25-29 2003.

[168] V. Nallur, R. Bahsoon, and Xin Yao. Self-optimizing architecture for ensuring quality attributes in the cloud. In *Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture. WICSA/ECSA 2009.*, 2009.

[169] David Oppenheimer, Brent Chun, David Patterson, Alex C. Snoeren, , and Amin Vahdat. Service placement in a shared widearea platform. In *USENIX Annual Technical Conference*, page 273–288, 2006.

[170] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May-June 1999.

[171] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M.P. Frank, and C. Chokkareddy. Ocean: the open computation exchange and arbitration network, a market approach to meta computing. In *Parallel and Distributed Computing, 2003. Proceedings. Second International Symposium on*, pages 185 – 192, oct. 2003.

[172] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review*, 41(3):289–302, 2007.

[173] M. Parashar and S. Hariri. Autonomic computing: An overview. In *Proceedings of the Workshop on Unconventional Programming Paradigms*, volume 3566 of *In Lecture Notes in Computer Sciences*, pages 247–259. Springer Verlag, 2005.

[174] Rob Powers, Moises Goldszmidt, and Ira Cohen. Short term performance forecasting in enterprise systems. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 801–807, New York, NY, USA, 2005. ACM.

[175] M. Randles, D. Lamb, and A. Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 551 –556, 2010.

[176] Diana Richards and Jude C. Hays. Navigating a nonlinear environment: An experimental study of decision making in a chaotic setting. *Journal of Economic Behavior & Organization*, 35(3):281–308, 1998.

[177] Étienne Rivière, Roberto Baldoni, Harry Li, and José Pereira. Compositional gossip: a conceptual architecture for designing gossip-based applications. *ACM SIGOPS Operating Systems Review*, 41(5):43–50, October 2007.

[178] Mema Roussopoulos and Mary Baker. Practical load balancing for content requests in peer-to-peer networks. *Distributed Computing*, 18(6):421–434, June 2006.

[179] P. Ruth, P. McGachey, and null Dongyan Xu. Viocluster: Virtualization for dynamic computational domains. In *IEEE International Conference on Cluster Computing*, 2005.

[180] Thomas L. Saaty. The seven pillars of the analytic hierarchy process. In *Proceedings International Conference on Multiple Criteria Decision Making*, volume 507 of *Lecture notes in economics and mathematical systems*, pages 15–37. Springer, Berlin, 2001.

[181] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Using aggregation for adaptive super-peer discovery on the gradient topology. In *Self-Managed Networks, Systems, and Services*, volume 3996 of *Lecture Notes in Computer Sciences*, pages 73–86. Springer, 2006.

[182] Jan Sacha and Jim Dowling. A gradient topology for master-slave replication in peer-to-peer environments. In *Databases, Information Systems, and Peer-to-Peer Computing*, volume 4125 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2007.

[183] S. M. Sadjadi and P. K. McKinley. A survey of adaptive middleware. Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, 2003.

[184] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):42, May 2009.

[185] Christoph Schroth and Till Janner. Web 2.0 and soa: Converging concepts enabling the internet of services. *IT Professional*, 9(3):36–41, May/June 2007.

[186] B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, Sept.–Oct. 2003.

[187] Kai Shen. Structure management for scalable overlay service construction. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, San Francisco, California, March 29-31 2004.

[188] Kai Shen, Hong Tang, Tao Yang, and Lingkun Chu. Integrated resource management for cluster-based internet. In *5th Symposium on Operating Systems Design and Implementation*, 2002.

[189] Kai Shen, Tao Yang, and Lingkun Chu. Cluster load balancing for fine-grain network services. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, 2002.

[190] Jeffrey Shneidman, Chaki Ng, David C. Parkes, Alvin AuYoung, Alex C. Snoeren, and Amin Vahdat Brent Chun. Why markets could (but don't currently) solve resource allocation problems in systems. In *Tenth Workshop on Hot Topics in Operating Systems*, 12–15 June 2005.

[191] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. *Computer Communications*, 31(2):402–412, 2008.

[192] H.A. Simon. A behavioral model of rational choice. *The Quarterly Journal of Economics*, 69(1):99–118, 1955.

[193] R. Sterritt. Autonomic computing. *Innovations on Systems and Software Engineering*, 1(1):79–88, 2005.

[194] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. In *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, pages 58 –67, sep 1994.

[195] G. Tesauro, D.M. Chess, W.E. Walsh, R. Das, A. Segal, I. Whalley, J.O. Kephart, and S.R. White. A multi-agent systems approach to autonomic computing. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

[196] Gerald Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22 –30, jan.-feb. 2007.

[197] Leigh Tesfatsion. Agent-based computational economics: Growing economies from the bottom up. *Artificial Life*, 8(1):55–82, Winter 2002.

[198] H. Tianfield. Multi-agent based autonomic architecture for network management. In *Proceedings. IEEE International Conference on Industrial Informatics (INDIN)*, pages 462– 469, 2003.

[199] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[200] P. Van Roy, A. Ghodsi, S. Haridi, J.B. Stefani, T. Coupaye, A. Reinefeld, E. Winter, E.P. Mobilfunk, and R. Yap. Self management of large-scale distributed systems by combining peer-to-peer networks and components. Technical report, CoreGrid, 2006.

[201] Ymir Vigfusson, Ken Birman, Qi Huang, and Deepak P. Nataraj. Optimizing information flow in the gossip objects platform. *SIGOPS Oper. Syst. Rev.*, 44:71–76, April 2010.

[202] S. Voulgaris. *Epidemic-Based Self-Organization in Peer-to-Peer Systems*. PhD thesis, VU University, Amsterdam, Netherlands, 2006.

[203] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[204] Spyros Voulgaris and Maarten van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 1143–1152. Springer Berlin / Heidelberg, 2005.

[205] N.J. Vriend. A new perspective on decentralized trade. *Economie Appliquée*, 47(4):5–22, 1994.

[206] W3C. Web services architecture. Electronic Web Publication, 2004.

[207] C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, and W.S. Stornetta. Spawn: a distributed computational economy. *Software Engineering, IEEE Transactions on*, 18(2):103 –117, feb 1992.

[208] Kent D. Wall. A model of decision making under bounded rationality. *Journal of Economic Behavior & Organization*, 20(3):331–352, April 1993.

[209] W.E Walsh, G. Tesauro, J.O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 70–77, 2004.

[210] W.E. Walsh and M.P. Wellman. A market protocol for decentralized task allocation. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 325 –332, jul 1998.

[211] Michael P. Wellman. *Market-based control: A paradigm for distributed resource allocation*, chapter Market-oriented programming: some early lessons, pages 74–95. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

[212] Matt Welsh and David Culler. Adaptive overload control for busy internet server. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and System*, 2003.

[213] Rich Wolski, Neil Spring, and Jim Hayes. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 15(1):757–768, Octuber 1999.

[214] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: A survey. In *Intelligent Agents*, volume 890 of *Lecture Notes in Computer Science*, pages 1–39. 1995.

[215] Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2004.

[216] Lingyun Yang, I. Foster, and J.M. Schopf. Homeostatic and tendency-based cpu load predictions. In *Proceedings. International Parallel and Distributed Processing Symposium.*, page 9, 2003.

[217] F Ygge and H Akkermans. Decentralized markets versus central control: A comparative study. *Journal of Artificial Intelligence Research*, 11(11):301–333, 1999.

[218] 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.

[219] Ying Zhang and M. Fromherz. Message-initiated constraint-based routing for wireless ad-hoc sensor networks. In *First IEEE Consumer Communications and Networking Conference*, pages 648–650, January 5–8 2004.

[220] Huican Zhu, Hong Tang, and Tao Yang. Demand-driven service differentiation in cluster-based network servers. In *Proceedings Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 679 –688 vol.2, 2001.

[221] S. Zilberstein. Models of bounded rationality. In *AAAI Fall Symposium on Rational Agency*, 1995.