UNIVERSITÀ DEGLI STUDI DI TRIESTE
Sede Amministrativa del Dottorato di Ricerca

Posto di dottorato attivato grazie al contributo di ES.TEC.O.

XX CICLO DEL
DOTTORATO DI RICERCA IN
INGEGNERIA DELL'INFORMAZIONE

Performance Control of Internet-based Engineering Applications

(Settore scientifico-disciplinare ING-INF/05)

DOTTORANDO
Paolo Vercesi

COORDINATORE DEL COLLEGIO DEI DOCENTI
Chiar.mo Prof. **Alberto Bartoli**
Università degli Studi di Trieste
FIRMA: .....

RELATORE
Chiar.mo Prof. **Alberto Bartoli**
Università degli Studi di Trieste
FIRMA: .....

Contents

Acknowledgments

I would like to express my sincere gratitude to all the people who helped me during these years. First of all, I feel sincerely grateful to my tutor Prof. Alberto Bartoli. I want to express my thanks to him for all the efforts lavished in motivating me and in addressing my work toward interesting scientific topics.

A special thanks is for my colleagues: Cyril and Eric, despite working on different topics we shared many worthwhile discussions in the laboratory and in front of countless coffees. It is always nice to speak with them.

This thesis has been made possible thanks to the financial support of Esteco, besides this I want to thank Prof. Carlo Poloni and Luka Onesti for their unlimited and unreserved understanding. Finally, I would like also to thank my friends, all the people working at Esteco and all the people who belonged to the "Laboratorio di programmazione delle reti di calcolatori" in these three years. Last but not least, a heartfelt thanks to my family, for its lovely and full support.

Abstract

Thanks to technologies able to simplifying the integration among remote programs hosted by different organizations, engineering and scientific communities are embodying service oriented architectures to aggregate, share and distribute their computing resources to process and manage large data sets, and to execute simulations through Internet. Web Service, for example, allow an organization to expose the functionality of its internal systems on the Internet and to make it discoverable and accessible in a controlled manner.

Such a technological advance may enable novel applications also in the area of design optimization. Current design optimization systems are usually confined within the boundary of a single organization or department. Modern engineering products, on the other hand, are assembled out of components developed by several organizations. Composing services from the involved organizations, a model of the composite product can be described by an appropriate workflow. Such composite service can then be used by a inter-organizational design optimization system.

The design trade-offs that have been implicitly incorporated within local environments, may have to be reconsidered when deploying these systems on a global scale on the Internet. For example: *i)* node-to-node links may vary their service quality in an unpredictable manner; *ii)* third party nodes retains full control over their resources including, e.g., the right to decrease the resource amount temporarily and unpredictably.

From the point of view of the system as a whole, one would like to maximize the performance, i.e. throughput the number of candidate design evaluations performed per unit of time. From the point of view of a participant organization, however, one would like to minimize the cost associated with each evaluation. This cost can be an obstacle to the adoption of this distributed paradigm, because organizations participating in the composite service share they resources (e.g. CPU, link bandwidth and software licenses) with other, potentially unknown, organizations. Minimizing such cost while keeping performance delivered to clients at an acceptable level can be a powerful factor for encouraging organi-

zations to indeed share their services.

The scheduling of workflow instances in such a multi-organization, multi-tiered and geographically dispersed environment have strong impacts on performance. This work investigates some of the fundamental performance and cost related issues involved in such a novel scenario. We propose an adaptive admission control to be deployed at the workflow engine level that limits the number of concurrent jobs. Our proposal can be implemented very simply: it handles the service as black-boxes, and it does not require any hook from the participating organizations.

We evaluated our technique in a broad range of scenarios, by means of discrete event simulation. Experimental results suggest that it can provide significant benefits guaranteeing high level of throughput and low costs.

Estratto

Grazie alle tecnologie capaci di semplificare l'integrazione tra programmi remoti ospitati da differenti organizzazioni, le comunità scientifica ed ingegneristica stanno adottando architetture orientate ai servizi per: aggregare, condividere e distribuire le loro risorse di calcolo, per gestire grandi quantità di dati e per eseguire simulazioni attraverso Internet. I *Web Service*, per esempio, permettono ad un'organizzazione di esporre, in Internet, le funzionalità dei loro sistemi e di renderle scopribili ed accessibili in un modo controllato.

Questo progresso tecnologico può permettere nuove applicazioni anche nell'area dell'ottimizzazione di progetti. Gli attuali sistemi di ottimizzazione di progetti sono di solito confinati all'interno di una singola organizzazione o dipartimento. D'altra parte, i moderni prodotti manifatturieri sono l'assemblaggio di componenti provenienti da diverse organizzazioni. Componendo i servizi delle organizzazioni coinvolte, si può creare un *workflow* che descrive il modello del prodotto composto. Questo servizio composto può a sua volta essere usato da un sistema di ottimizzazione inter-organizzazione.

I compromessi progettuali che sono implicitamente incorporati per architetture locali, devono essere riconsiderati quando questi sistemi sono messi in opera su scala globale in Internet. Ad esempio: *i)* la qualità delle connessioni tra i nodi può variare in modo impredicibile; *ii)* i nodi di terze parti mantengono il pieno controllo delle loro risorse, incluso, per esempio, il diritto di diminuire le risorse in modo temporaneo ed impredicibile.

Dal punto di vista del sistema come un'entità unica, si vorrebbero massimizzare le prestazioni, cioè, per esempio, il *throughput* inteso come numero di progetti candidati valutati per unità di tempo. Dal punto di vista delle organizzazioni partecipanti al *workflow* si vorrebbe, invece, minimizzare il costo associato ad ogni valutazione. Questo costo può essere un ostacolo all'adozione del paradigma distribuito, perché le organizzazioni partecipanti condividono le loro risorse (cioè CPU, connessioni, larghezza di banda e licenze software) con altre organizzazioni potenzialmente sconosciute. Minimizzare questo costo, mentre si mantengono le prestazioni fornite ai clienti ad un livello accettabile, può essere un potente fat-

tore per incoraggiare le organizzazioni a condividere effettvivamente le proprie risorse.

Lo *scheduling* di istanze di *workflows*, ovvero stabilire quando e dove eseguire un certo *workflow*, in un tale ambiente multi-organizzazione, multi-livello e geograficamente disperso, ha un forte impatto sulle prestazioni. Questo lavoro investiga alcuni dei problemi essenziali di prestazioni e di costo legati a questo nuovo scenario. Per risolvere i problemi inviduati, si propone un sistema di controllo dell'accesso adattativo davanti al *workflow engine* che limita il numero di esecuzioni concorrenti. Questa proposta può essere implementata in modo molto semplice: tratta i servizi come *black-box* e non richiede alcuna interazione da parte delle organizzazioni partecipanti.

La tecnica è stata valutata in un ampio spettro di scenari, attraverso simulazione ad eventi discreti. I risultati sperimentali suggeriscono che questa tecnica può fornire dei significativi benefici garantendo alti livelli di *throughput* e bassi costi.

Chapter 1

Introduction

Thanks to Internet technologies, engineering and scientific communities are embodying service oriented architectures to aggregate, share and distribute their computing resources in order to process and manage large data sets [7], and to execute simulations through Internet [40, 59, 54, 64]. Many examples come from the workflow management and scientific computing areas. The scientific community is also adopting the workflow technology and it is facing the challenges of the scientific workflows [31]. The engineering community is testing and using this novel paradigm also [54].

1.1 Scenario and Motivations

The widespread diffusion and acceptance of protocols for programmatic interactions across different organizations connected to the Internet (e.g. web services, and grid computing), along with the plenty of software implementing these protocols, has made it feasible the *composition* of services across the Internet [19, 55, 60, 9]. Services can be aggregated together, even when exported by different and remote organizations, to compose workflows that provide streamlined functionality. A service participating in a composite service could be, in turn, a composite service described by its own workflow.

In this thesis, issues related to Internet-based composition of services for *design optimization* are explored. The work of this thesis has been funded with a scholarship offered by Esteco (Engin Soft Tecnologie per l'Ottimizzazione). Esteco (`http://www.esteco.com`) is an Italian company founded in 1999 to transfer the knowledge acquired by its founders while working on an European Union sponsored project on design optimization (FRONTIER) into a successful commercial product. Esteco turned a research-stage product into a world-class, industrial-strength, multi-objective optimization platform: modeFRON-

TIER. modeFRONTIER is a multi-objective optimization and design environment, designed to enable easy coupling to almost any computer aided engineering (CAE) tool whether commercial or in-house. It provides an environment which allows product engineers and designers to integrate their various CAE tools, such as CAD, finite element structural analysis and computational fluid dynamics (CFD) software. Using a variety of state-of-the-art optimization techniques, ranging from gradient-based methods to genetic algorithms, the process or design of interest can be optimized by specifying objectives and defining variables which affect factors such as geometric shape and operating conditions. In practice, modeFRONTIER becomes a wrapper around the CAE tool, performing the optimization by modifying the value assigned to the input variables and monitoring the outputs. So far, the software modeFRONTIER has been widely used in different application fields such as: aerospace, appliances, bioengineering, pharmaceutics, civil engineering, manufacturing, marine engineering, crash tests, structural analysis, vibro-acoustics and turbo-machinery.

The work in this thesis is motivated by the observation that design optimization is one of the key problems to be faced in nearly all industrial sectors. Optimizing, or simply improving, the design of a component or a system is a complex and time-consuming job, but it is also a necessity to stay competitive. Satisfying this requirement while achieving faster product development times, faster product innovation and turnaround, lower costs is becoming more and more difficult.

In many cases, the use of specialized simulation software has become the only practical way for addressing such conflicting goals. In particular, *simulation-based optimization* systems [62, 8, 6], and *design-optimization* environments allow exploring efficiently the design search space: they generate a set of candidate designs, evaluate each design by means of simulation and then generate a new set of better designs, based on specific search optimization strategy. The simulation techniques used for evaluating each "virtual prototype" include disparate applications. The entire process proceeds automatically and iterates until one or more satisfactory designs are found. At this point, the results are given to human specialists, e.g., a team of engineers, for further analysis and evaluation.

This approach is now widely used and has become a key component of many CAD/CAE tools. Concerning the specific system with which are more familiar [27], significant successful applications have been performed in a number of different sectors: aerospace & defense [29], appliances, architectural, automotive [28], biomedical [63], experimental data, food & beverage, manufacturing, marine & off-shores [48, 14], turbo-machinery and casting [35]. Other applications can be found in multi-disciplinary design optimization [30, 13].

Recent developments in software technology have opened a new range of possibilities for design optimization systems. Technologies capable of greatly simpli-

fying the integration among remote programs hosted by different organizations are now widely available. Web service technology [32, 61, 4], for example, allows an organization to expose (some of) the functionality of its internal systems on the Internet and to make it discoverable and accessible through the World Wide Web in a controlled manner. Multiple organizations can be involved in the simulation of a candidate design, by composing their respective simulation modules on the Internet. It has thus become possible to build *inter-organizational* services by combining multiple services exported by single organizations. As a result, one can build complex services resulting from the Internet-based integration of simpler building blocks, possibly consisting of simple adapters wrapped around existing systems.

Contemporary design optimization systems are usually confined within the boundary of a single organization. Modern engineering products, on the other hand, are assembled out of components developed by several organizations. One may devise a scenario in which each organization makes available on the Internet services for simulating its own components and, based on such services, an inter-organizational service that simulates the behavior of the whole product is constructed. Such a composite service could then be used by a design optimization system. This approach could lead to more accurate results, simplify the tailoring of generic designs to specific scenarios, allow identifying fundamental design problems earlier, make it easier the management of product upgrades and so on.

Engineering and scientific workflows are the basis for this scenario. Current research on *scientific workflows* is aimed at creating a *science of workflows* [31]. More specifically scientific workflows and service oriented computing research are oriented toward service semantics and coherent service composition, for example focusing on: reconfigurable architectures, end-to-end security, infrastructures for data and processing integration, automated semantical service discovery, analysis of composability for replaceability, dynamic and adaptive processes, QoS aware service composition, business-driven automated composition, autonomous (self-configuring, self-adapting, self healing, self optimizing, self protecting) management services, service applications engineering, flexible gap-analysis techniques, service versioning, adaptivity and governance [52].

This thesis is concerned with *performance issues* related to integration of engineering services. The fundamental design trade-offs that have been incorporated, perhaps implicitly, within design optimization systems for local environments, may have to be reconsidered when such systems are deployed on a global scale, or are integrated with similar systems distributed geographically on the Internet. The Internet affects distributed systems in multiple ways: *i)* node-to-node links are subject to Internet traffic and, consequently, they may vary their service quality in an unpredictable manner; *ii)* third party nodes may share their

computing resources with other organizations, while retaining full control over those resources including, e.g., the right to decrease their amount temporarily and unpredictably.

Motivated by the above considerations, this thesis explores such novel applications of simulation by focusing on some of its fundamental performance-related issues. For example, a system involving several remote organizations can be managed in a centralized way, by an engine that moves data back and forth among all modules involved, or in a distributed way, with data flowing from one module to the next. The former is much simpler to implement, but the latter may be much more efficient depending on the bandwidth available and the size of the data. Gaining insight in this area is crucial for understanding how to structure design optimization systems spanning multiple organizations.

There are also fundamental questions to be answered regarding the desirable trade-offs. For example, from the point of view of the system as a whole, one would like to maximize throughput, i.e., number of number of candidate designs evaluated per unit of time. From the point of view of a participant organization, however, one would like to minimize the execution time of its services. These points of view may conflict because such services are usually associated with expensive software licenses that cannot be held by more than one process at a time and because an organization usually has many internally generated jobs that need such services. It follows that overlapping communication with computation at a node, a simple strategy for improving throughput, may lead to longer times in which a license is kept busy, thereby subtracting precious resources for the execution of internally generated jobs.

1.2 Contribution

In this thesis, autonomous mechanisms and policies for controlling the scheduling of jobs in such a highly dynamic environment are proposed. The main goal is the minimization of the resource usage at the participating organizations while maintaining the performance delivered to clients at an acceptable level. To be feasible and attractive, solutions cannot require major modification to current organization computing infrastructure, organizations and their services must be threated as black-boxes.

The approach proposed in this thesis consists of a form of admission control at the entrance point of the *application workflow* that is simple to deploy in practice and does not need any hook from the participating organizations. The maximum number of jobs that can be injected within the grid of services is simply varied dynamically, based on performance measures taken on line.

This thesis proposal can be implemented very simply, the proposed admission control places an upper bound on the multiprogramming level and it delays excess

jobs. The number of jobs that can be injected into the composite service is varied dynamically with an adaptation policy driven by the current estimates of latency and throughput for the composite service as a whole. The key feature of this approach is that it does not require any hook from the participating organizations and it treats the entire workflow as a black-box. Furthermore this solution does not affect in any way current web service middleware, it requires only to expose the composite service as a regular service and to interpose the admission control module between the composite service entry point and the workflow engine.

The technique has been evaluated in a broad range of scenarios, by means of a detailed event driven simulator. This proposal is indeed capable of finding automatically a suitable trade-off between throughput and resource usage, even in such a dynamic scenario. Experimental results suggest that the proposal can indeed provide significant benefits to service providers.

Chapter 2

Literature Overview

The work of this thesis shares some similarities with other works in the following fields: Grid Workflows, Parallel Computing, Transaction Processing Systems, Services and Applications on the Web, Computer Systems Control and Service Composition Virtualization.

2.1 Grid Workflows

A large number of research papers address the question of mapping sets of tasks onto sets of processors in a view to minimizing overall makespan. Many of these papers address the case where tasks are independent. Several of the recent papers in this field also take into account data storage issues.

In [18] they consider parameter sweep applications executed on the Grid. A parameter sweep application is a set of n independent sequential task, each task works on a set of input files and produces one output file, the task structure is no further detailed. In order to minimize the overall makespan (i.e. the time between the first input files is sent to a computational server and the last output file is returned to the user), they integrate existing scheduling heuristics into a general adaptive scheduling algorithm.

While makespan minimization is also one of our objectives, there are some differences with our work: *i)* each sweep application task is assigned to a specific host, they do not consider assigning subtasks to other hosts, *ii)* all tasks are independent, in contrast in our domain the parameter configurations depend on the results of previous tasks, *iii)* the scheduler need to know the status of all computing and communication resources, *iv)* in addition to makespan minimization we also try to minimize the cost incurred by the organizations participating in the grid, *v)* to account for the Grid dynamic nature the plan refinement is called repeatedly with a period fixed by the scheduler itself while in our approach the

period is not fixed. Furthermore they require the complete and precise knowledge of *i)* current topology of the Grid (i.e. number of clusters, number of hosts in those clusters, network and CPU loads), *ii)* the number and location of copies of all input files, and *iii)* the list of computations and the transfers currently underway or already completed.

Large scale applications expressed as a set of tasks with data dependencies between them are known as application workflows. In [59] they reduce application workflows completion time not only by finding optimal task allocation but also considering the overhead induced by the workflow engine. The experiments, they performed, have been done in a stable execution environment, resources have been provisioned in advance and no alteration affected the running system. The proposed system is manually tuned and cannot adaptively react to resource load or availability changes.

In [64] they consider scientific workflow systems for analysis and processing of large data sets. They propose an optimized run-time support system to support scientific data-intensive workflow execution on a distributed environment. The system requires a persistent data storage manager coordinating data between nodes, the workflow meta-data manager is always aware about data placement.

In [54] they describe the ASP system, a testbed based on Web Services for coupled multi-physics simulations. Their results suggest that potential performance bottlenecks identified in the literature may not be major issues in practice, and that a standards-compliant implementations can delivery excellent scalable performance even on coupled problems. They found that previous studies on SOAP and XML, reporting that the use of SOAP and XML without sophisticated optimizations impose a large performance penalty in scientific applications, are misleading. They do not address the scheduling of composite web service nor the conflicts arising when running multiple instances of the same service.

In [20] they investigate build time and runtime issues related to decentralized orchestration of composite web services. They improve the system throughput by using a decentralized execution approach, but they do not use any type of scheduling nor admission control.

2.2 Parallel Computing

The effect of limited network resources on parallel applications has been recently studied in [39]. They determined the network-related properties of different distributed computing applications: *i)* how the performance of an individual application is influenced by the network resources (e.g. link bandwidth and latency); *ii)* how traffic generated by individual applications affect the state of the network resources (e.g. packet-drop rate and RTT). They considered canonical (i.e. LU Decomposition, Task Allocation, Jigsaw Puzzle and N Queen problem) par-

allel applications using message passaging interface running on a WAN. In our work we investigated the effect of limited link bandwidths on the performance of application workflows and in spite of the differences between the considered applications we found similar results.

2.3 Transaction Processing Systems

Adaptive algorithms for setting an optimal multi programming level in transaction processing systems and database management system has been studied since a long time. The works described in this section contemplate a non-mutable set of resources and they adapt the multi-programming level considering the current load. Multi programming limiting is almost always implemented by an admission control module.

A locking-based concurrency control algorithm to avoid the trashing problem has been described in [15]. The algorithm determines when (and when not) to admit new transactions into the system based on its knowledge of the current system state. This algorithms operates almost without any assumption on the system but the number of expected locks for each transaction.

In [34] they prevent trashing in transaction processing by controlling the number of concurrently running transactions. Because the optimal concurrency level strongly depends on the workload characteristics which may change in time, they propose two adaptive algorithms the adjustment of an upper bound for the concurrency level.

As in our work they are not concerned about any internal details of the system, they are solely interested in the functional relationship of the concurrency level n as the input and the resulting performance P as the output of the system. The throughput T is used as the performance index P. However they point out that alternative quantities with similar shape are eligible. In particular they assume that this function $P(n)$ at each time has a shape that is monotonically increasing up to a maximum at n_{opt}, and then decreasing. In other words, they assume the existence of a local maximum that is also a global one.

They assume performance to be a function of the time allowing for almost arbitrary changes of the load characteristics. They also require the dynamic behavior to have some locality in the sense that the shape of the curve at time t_i is a good estimate for its shape at time t_{i+l}.

They informally describe the algorithm as follows: starting at time $t = 0$ with an arbitrary load value, the algorithm has to find the "ridge" of the "mountain" and to track it along the time axis. Additionally, they do not know the shape of the mountain in total but all information we can obtain is the series of realized load/performance pairs from the past.

They studied also the problem of selection of the sample interval amplitude, it

should be large enough to allow an accurate estimation of the relevant quantities and should be small enough to allow the assumption of stationary behavior. Because our control module is practically self-clocked, in our work we do not need to address this problem.

In [49] they present an adaptive distributed middleware for data replication that is able to adjust to changes in the amount of load submitted to the different replicas and to the type of workload submitted. It combines load-balancing techniques with feedback drive adjustments of multiprogramming levels. Current applications cannot be described with static parameters, at each replica they use the same parabola approach described in [34] and they implement a load balancing between replicas

Differently from the other works in [58] they set a lower bound, instead of an upper bound, for the multiprogramming limit this strategy is necessary to render more effective the external scheduling of the queued DBMS transactions. They maintain the most of the transaction in a queue that can be ordered by the application. So the application is free to schedule the queued transactions in accord with its preferences.

The performance of external scheduling are strongly affected by the multiprogramming limit i.e. the number of concurrent transaction processed by the DBMS. If the multiprogramming limit is too low, throughput will suffer, since not all DBMS resources will be utilized. On the other hand, if the multiprogramming limit is too high, there is insufficient control on scheduling.

The question of how to adjust the multiprogramming limit to achieve both goals simultaneously is an open problem, not just for databases but in system design in general. They studied the problem in the context in a transactional workload using both simulation and theoretical techniques. They develop a feedback based controller, augmented by queuing theoretic models for automatically adjusting the multiprogramming limit. In the controller they use a model for jump-start to the near-optimal value and then they control the system with small variations.

2.4 Services and Applications on the Web

In [25] they consider the execution of composite web services and they compare the throughput and the response time obtained by the following scheduling policies: Processor Sharing, transparent admission control (FIFO) and Shortest Job First (and other scheduling policies). With the transparent admission control policy they need to measure the provider overload, but in that work they do not specify how they evaluate such overload, furthermore admission control function is performed by a proxy that is tightly couple with the application server.

In a first part, using traces from the TPC-App benchmark [65] they investigate

the possibility of job size estimation for database centric Web Services. Such job size estimation capability is essential for the operation of the SJF algorithm. In a second part they study the impact of SJF scheduling on a standalone Web Service considering throughput and average response time as main performance metrics.

In papers considering interactive web services [45, 3, 21, 73, 26, 57] the response time is the only considered performance metric. They usually tend to guarantee a good response time in presence of overload, this good response time is not the minimum attainable but a value fixed by a service level agreement or a value determined by relative differences between different service classes. In general these works address the *overload control* problem.

In particular, in [21] they perform session based overload control for http requests. They have proposed and evaluated a scheduling algorithm which discriminates the scheduling of requests on the basis of the probability of completion of the session that the requests belong to.

In [73] they present a set of techniques for managing overload in complex, dynamic interactive Internet services. These techniques are based on an adaptive admission control mechanism that attempts to bound the 90_{th}-percentile response time of requests flowing through the service. This is accomplished by internally monitoring the performance of the service, which is decomposed into a set of event-driven stages connected with request queues. They fix a response time target and they focus on the response time of the applications and not on the overall throughput that is limited by the link bandwidths.

A method for admission control and request for scheduling for multiply-tiered e-commerce web sistes is presented in [26]. They achieve both stable behavior during overload and improved response times. Their proposal does not require any modifications to the host operating system, web server, application server or database, it can be implemented on a proxy. Their admission control requires two estimates: the load that a particular job will impose to the system and the capacity of that system. Load is estimated by recent system observations, to estimate the capacity of the system they use the method of incremental step described in [34]. Changes in the system hardware require system capacity recomputation. There is no dynamic adaption in this system.

2.5 Computer Systems Control

Feedback control theoretic model of web server are used in [45, 3, 57] to achieve overload protection, performance guarantees and service differentiation. Such control theoretic approaches require definition of the system model and identification of its parameters, on the other hand they provide guarantees of robustness and stability.

In contrast to the absolute guarantee and to the best effort differentiation. in [45] they propose an architecture to support the proportional differentiated service model for a Web server. With proportional differentiation performance level of differentiated classes maintain always the same proportion. They formulate the adaptive resource allocation problem as one of feedback control and apply feedback control theory to develop the resource allocation algorithm. The controller implementation require modification of the system subjected to control and the identification of the system itself modeled as a second order difference equation.

In [3] they describe performance control of a Web server using classical *feedback control theory*. They describe how to model a general Web server for purposes of performance control, present the equivalents of sensors and actuators, formulate a simple feedback loop, describe how it can leverage on real-time scheduling and feedback-control theories to achieve per-class response-time and throughput guarantees. In [57] they investigate a nonlinear discrete-time modeling of a Web server system. They develop and validate a control theoretic model for the admission control of a general single server queue. Their measurements in the laboratory setup show the robustness of the implemented controller, and how it corresponds to the results from the theoretical analysis and the simulations.

Control theoretic approaches to other software systems are presented in [53, 2, 38, 74, 11]. In particular in [53] they describe a methodology for designing controllers that manipulate tuning parameters of software system (e.g., an email server). They base identification on a statistical model fitting to historical measurements of the target being controlled. And they apply this methodology to a Lotus Notes groupware server to maintain a reference queue length to the desired level.

In [2] they describe an online control framework to design self-managing distributed computing systems that continually optimize their performance in response to changing computing demands and environmental conditions. They use an online control technique in conjunction with predictive filters to tune the performance of individual system components based on their forecast behavior. The system react to varied environmental and operating conditions by applying an online model learning online. They assume that the new parameters affecting component behavior are known, and their values are measurable. Then they estimate the incremental changes to the model due to the new environmental parameters.

In [38] they explore the properties that are necessary for performance model estimation of black-box computer systems when used together with adaptive feedback loops. They show that the method of least-squares estimation in conjunction with *Self Tuning Regulators* often gives rise to models that make the control loop perform the opposite action of what is desired. They propose extensions to con-

trollers that make them perform well, even when the model estimated would have degraded performance. Their approach is intended to help meet a *service level agreement* (SLA) goal (using latency as the metric) but only to get as close as possible to the SLA requirement without exceeding it, rather than maximizing performance.

In [74] they consider the use of artificial intelligence (AI) based solutions for reconfiguring distributed systems online to optimize for dynamically changing workloads. They present one approach to address the need to decide when and how to reconfigure. In particular, they learn to identify, from only low-level system statistics, which of a set of possible configurations will lead to better performance under the current unknown workload. This approach requires no instrumentation of the systems middleware or operating systems. They demonstrate that their adaptive configuration is able to outperform any single fixed configuration in the set over a variety of workloads, including gradual changes and abrupt workload spikes. They focus on the hardware of partitionable servers and they do not consider admission control or the multi programming level.

In [11] they recognize that efficient and robust data streaming are a critical requirement of emerging Grid applications, which are based on seamless interactions and coupling between geographically distributed application components. Furthermore the dynamism of Grid environments and applications requires that these services should be able to continually manage and optimize their operation based on system state and application requirements. They propose a self-managing data-streaming service based on online control strategies. The control strategy combines model-based limited *look-ahead* controllers [2] with rule-based managers to dynamically achieve adaptive behavior in Grid applications under various operating conditions. They also impose requirements for the live use of the results generated by the simulation workflow.

2.6 Service Composition Virtualization

In [43] a service bus is described, the main functionality of this bus is *virtualization*. This virtualization layer hides from a user of a service the implementation details of a service like the programming language used for its implementation, the hosting application server, the underlying operating system platform, and so on. Requests refer to the service interface, described in Web Service Description Language (WSDL) [72], the service bus virtualizes services based on interface description or semantic descriptions as well as based on non-functional properties. The service bus can optimize incoming requests according to various set of criteria, for example it might consider the workload of the overall environment, and the cost of mediating the request. To enable optimization the service bus must be able to retrieve information required for the various kind of optimizations like

state data and so on. Similarly, the service bus must be able to influence the state of services, e.g. it should be able to restart a certain service. For this purpose, services have to support corresponding interface in addition to the interface providing the proper (application) functions. Data that is providing the context for performing a request offered by a service is referred to as a WS-Resource [22]. An element of this data context is called a "resource property".

2.7 Grid Computing and Scientific Workflows

In engineering, assembling components into a compound product is a well known and ubiquitous practice. Software systems are built on reusable components too while workflows are composed of services available in the Internet. Services can provide both computing and storage resources, and they are the fundamental bricks of Grid applications. Thanks to the standardization efforts and to web services related technologies [32, 72, 50], interoperability across heterogeneous components is no more an issue. Anyway this work is not concerned with any specific component or service technology, thus conclusion and results have a general and broad applicability. Nor this work is concerned with languages for service composition nor with systems for discovering or aggregating services, those are already active research fields [31].

Grid computing has established itself as the dominant paradigm for wide-area high-performance distributed computing [11]. As Grid technologies and testbeds mature, they are enabling a new generation of scientific and engineering application formulations based on seamless interactions and couplings between geographically distributed computational, data, and information services. Simulation codes generate large amounts of data, which must be streamed efficiently and effectively between the distributed components.

Chapter 3

Design Optimization

Design optimization plays a fundamental role in any engineering product design and it has been rendered effective by use of computer aided tools. Design optimization frameworks or PIDO (Process Integration and Design Optimization) tools provide engineers with a set of building blocks to enable the creation of sophisticated workflows to integrate all the software components of a design process, as well as a set of state-of-the-art algorithms and extensive post-processing and decision support capabilities.

3.1 Introduction

The use of PIDO tools is gaining in popularity as a way to automate and drive simulation-based design processes. Product design and development rely heavily on computer aided engineering (CAE) software, such as tools for design (CAD), Finite Element Analysis (FEA), Computational Fluid Dynamics (CFD), and even proprietary software written within organizations to address particular needs. Usually these analysis are run separately from each other; even in the case where the result of one simulation is to be fed to another (an example would be running a CFD analysis to provide thermal boundary conditions for an FEA analysis) there is often a manual process of extracting the data of interest, and passing the file to the user who will perform the next step of the process.

Companies are now recognizing that such procedures should be automated in a way which allows all CAE tools to be run without human intervention, and provides for the transfer of the necessary data files between the components. This is not always a trivial task: in today environment, product development is often a global activity, distributed among research and development centers in different countries, and even on different continents. It is not uncommon to have CAE

tools installed in multiple locations as well; hence process integration in such cases must involve seamless file transfers, using protocols already in place.

Once the chain of simulations has been automated, it does not take much extra effort to convert it into an optimization process. The user, having specified which design inputs are to be the variables, will also define certain goals, or objectives; for example, to maximize efficiency and minimize cost. Often in real life applications the goals may be conflicting, which brings us to the concept of trade-off: by keeping the objectives separate, multi-objective optimization frequently gives rise to a Pareto Frontier of designs, all of which can be considered to be candidates for the optimum.

Once objectives have been defined (e.g. minimize weight, maximize efficiency, etc), and an optimization algorithm selected, the optimization software will take over, searching for configurations which provide the best compromise between goals which may well be in conflict. By combining ease-of-use with powerful, state-of-the-art optimization algorithms and process integration techniques, the optimization software provides the user with an invaluable aid for product design and development.

3.2 Synopsis

Design optimization is an iterative process where each step is divided in two phases: the first phase is the evaluation of a set of proposed designs through simulation, the second phase is the generation of a new set of hopefully better designs. Each design represents a virtual prototype of an engineering product.

The particular strategy or optimization algorithm [56] for producing new designs based on the evaluations of prior attempts is irrelevant to this discussion. The only distinction we made is for population based algorithms, e.g. genetic algorithms, these algorithms require the parallel evaluation of a current population, within this algorithms we distinguish between steady and block algorithms. Said n the number of individuals in the population, steady algorithms synthesize a new individual as soon as the evaluation of another individual terminates, on the other side, block algorithms synthesize a new entire population only after the evaluation of the previous n individuals. Thus for a steady algorithms the number of individuals currently under evaluation is n, for block algorithms it is initially n and then decreases till 0.

The process terminates when a termination condition is met, e.g. when either a satisfactory design is found or a predefined maximum number of designs have been evaluated. Evaluation of each design is performed according to a *simulation workflow* representing the sequence of tasks necessary for the analysis of the virtual prototype. As shown in Figure 3.1 design evaluation starts from the design variables x and leads to the performance measure $y = f(x)$.

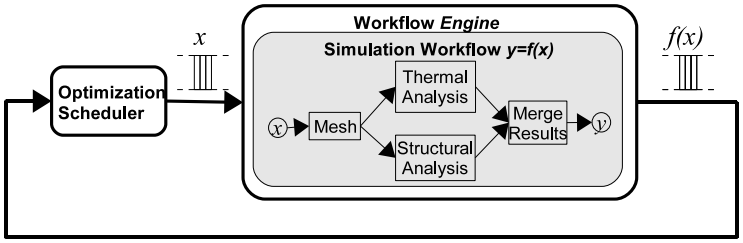

Figure 3.1: Simulation based optimization.

In the domain of our interest, design optimization systems are composed of two main functional blocks: the *optimization scheduler*, and the *workflow engine*. The optimization scheduler is responsible for the synthesis of new designs based on the results from the prior ones, it is also responsible for submitting evaluation requests to the workflow engine. The optimization scheduler schedules the evaluation requests in accord to the optimization algorithm. The workflow engine receives evaluation requests from the scheduler (or from any other client) and executes the simulation workflow with the specified data.

The simulation workflow describes which services are to be executed, in which order and on which data (see Figure 4.1: Composite service logic view.) In other words, the simulation workflow appears to its clients as a single composite service, internally composed by multiple services. A *job* consists of an execution of the composite service on a specified set of input data and it produces a set of output data.

The actual execution of jobs is orchestrated by the workflow engine, that invokes the relevant composing services as appropriate. This activity usually involves the movements of large amount of data. Both the optimization scheduler and the workflow engine are internal modules of the composite service that need not be exposed to clients. In the inter-organizational scenario, the composing services are exported by potentially different organizations. Communication between services and optimization machinery occurs through the Internet.

3.3 Strategies for Design Optimization

The optimization scheduler may submit new tentative designs for evaluation according to several different policies. We explored the two policies most commonly used, as follows. With the *block* submission policy the scheduler first generates a batch of n workflow instances (designs), submits all these instances to the workflow engine and waits for completion of all of them. The scheduler then generates a new batch based on the results of the previous batch, and so on. This submission pattern is typical of optimization schedulers based on generations of population such as genetic algorithms and evolutionary strategies [16]. With the

steady (or *steady-state*) policy the scheduler starts a batch of n instances and then submits a new evaluation as soon as a previous one has completed. Note that submission policies are largely independent of the specific search strategy implemented by the scheduler.

3.4 Engineering Workflows

Engineering products can be described by a parametric model, this parametric model has inputs and outputs values, those values can in general be of any type: real values, strings, arrays, multidimensional quantities or files, there is no any specific requirement on this aspect. On the other hand the optimization process works in general with numerical quantities, but this requirement is specific to the particular optimization algorithms, the most diffused algorithms: genetic algorithms, evolutionary strategies [56], gradient based algorithms, mixed integer algorithms work with numeric values.

The optimization process deal with *decision variables* and *goals*, decision variables define the design search space while goals define the objectives to obtain and the constraint to meet. In general input values are function of the decision variables while constraints and objectives are function of the decision variables and of the output values.

Depending on the required level of details and depending on the specific analysis, an engineering product can have more parametric models. The more advanced is the design stage the more detailed is the model. Detailed models take in account many physical aspect of the product, different analysis are conducted by different tools that must be integrated into the simulation workflow.

3.5 Inter-organizational Design Optimization

Many factors influence the performance of inter-organizational design optimization systems [67], in particular: node-to-node link quality, storage architecture, optimization strategy, and communication computation overlap pattern, all have a significant impact on performance. Other factors are the computing power at the computing nodes (i.e. number of CPUs), the number of available licenses for each solver, the number of concurrent evaluations allowed.

A crucial observation is that most of these factors are beyond the control of the entity building the inter-organizational design optimization system. Other factors cannot be known and cannot even be estimated (without significant intrusion into third party systems). Finally, nearly all of these factors may vary unpredictably during execution amount of resources available at each organization, bandwidth of the links, load imposed by other systems on the organizations and on the links, just to name a few.

Current design optimization systems do not suffer from these uncertainties, because the optimization is performed on local hardware (e.g. on a local cluster) and with dedicated resources fully allocated to the optimization task. In such a kind of environment manual fine tuning usually guarantees the best arrangement between the considered performance indexes. In summary, what makes this scenario challenging is that performance may heavily depends on a myriad of factors, that may be unknown, may vary unpredictably and are beyond the control of the scheduling machinery.

Chapter 4

Workflows and Service Composition

4.1 Definitions

Based on Workflow Management Coalition [75], a workflow is: "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*". Scientific and engineering workflows adhere to this definition although they represent engineering simulations or scientific calculations rather than business processes. Each workflow is defined by a workflow process definition also called *workflow schema* [1]. A schema specifies which application programs need to be executed, in what order and on which data. Each application program appears as a *service* that can be invoked by its clients [51]. A *workflow instance* or *job* consists of an execution of a schema on a specified set of input data and it produces a set of output data.

In other words a workflow is a set of tasks with data dependencies between them. Composite services can be internally modeled as workflows, i.e., workflows implemented by means of services exported by potentially different organizations connected to the Internet. For ease of discussion but without loss of generality the focus can be stressed on a single composite service exporting one single schema. *Input data* is composed of input parameters and input files, similarly *output data* is composed of output parameters and output files.

Usually, parameters are enclosed in the job whereas files are not. Files are uploaded/downloaded when needed, based on directives enclosed in the job. Execution of jobs is orchestrated by a *workflow engine*, that interprets a high-level description of a schema and invokes the relevant composing services as appro-

priate. This involves the movements of large amount of data over a wide-area network. Conceptually, the workflow engine is an internal module of the composite service that needs not be exposed to clients.

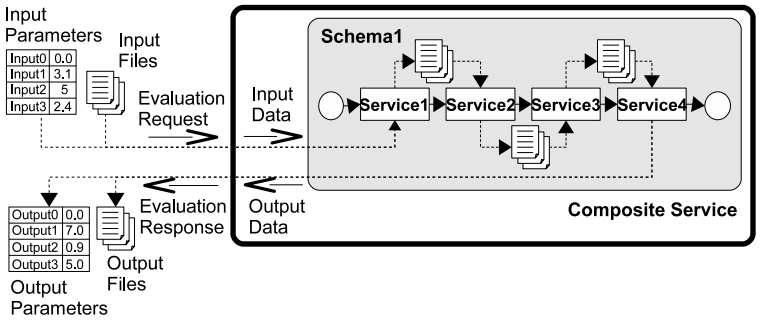

Figure 4.1: Composite service logic view.

Figure 4.1 shows the logic view of a typical composite service for the evaluation of a scientific workflow. The scientific workflow is defined by the workflow schema in the gray area, the schema is composed of four serially connected services. The schema specifies also the input data and the output data needed to carry out the workflow evaluation. Input and output data take the form of files and parameters. From the depicted schema files created by one service are used by the next service, so the next service needs to download the files while the previous service has to make available those files. The simulation workflow is wrapped by the composite service, that is a service like the others, but it masks the simulation workflow details, there is no limit to the recursion level.

4.2 Execution

The execution of workflow instances is performed by a *workflow engine*, that interprets a high-level description of a schema for example defined in the BPEL language [10]. Description of workflow schemes and their execution is usually performed under control of a workflow management system (WfMS). The client application requests the execution of a workflow to the workflow management system which instantiates a workflow engine to fulfill the client request. For the sake of this thesis the optimization scheduler plays the role of the client application.

This schema could be implemented as shown in Figure 4.2. Each service can be provided by a different organization, different organizations could correspond to different department or branches of the same organization are they can belong to different enterprises or institutions. Communication across organizations occur, in general, through the Internet, which is the case of main interest in this work, and not through dedicated links. Obviously, a given organization could

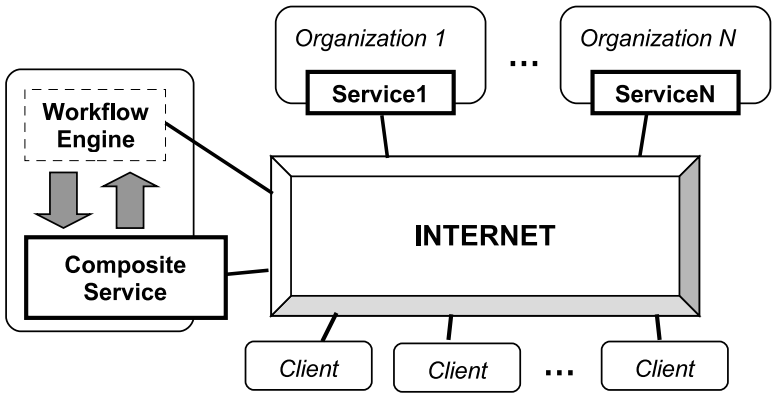

Figure 4.2: Grid workflow sample physical layout.

provide more services and similar, or functionally equivalent, services could be provided by more organizations [23, 47], this option is no further analyzed in this work. The figure does not explicitly shows the scheduler that can be represented by any of the clients, the workflow engine also communicate through the Internet.

The optimization scheduler is the closest process to the end users. An end user selects an optimization scheduler, feeds a simulation workflow to the optimization scheduler, and asks the scheduler to perform the optimization. In turn, the optimization scheduler submits workflow evaluations to the workflow management system.

Workflows are used in several fields of science and engineering, usually for elaborating data coming from experiments [12, 37, 46]. Although this thesis focuses on workflows for design optimization systems, much of our analysis may be applied also to such contexts, in particular, whenever the delay incurred for transferring data between services is comparable to the computation time spent in services (see Section 6.2.4).

4.3 Model Design

At the core of the simulation workflow there is a set of solvers. A *solver* is an application code specialized in a specific computer aided engineering task, e.g., computer aided design applications, finite element analysis solvers, computational fluid dynamics solvers or proprietary software. Often it is a commercial licensed software. A solver execution usually proceeds as follows:

1. it obtains an available license (the maximum number of solver instances that can be run simultaneously is bounded by the license agreement),

2. reads the input data,

3. performs the specific computation without any interaction with other entities, in particular, without interacting with other solvers,

4. and finally it produces the corresponding output data.

Input and output take the form of files. A solver is exposed to clients of an organization in the form of a *service*. A service takes care of all the necessary communication and synchronization activities, including the transfer of input/output data. Data is exchanged through a pull protocol, i.e., each service is responsible for downloading the data it needs. In most real-world settings, each server is tied to a specific solver.

4.4 Performance Metrics

Optimization latency is defined as the time required for completing the optimization session, *throughput* is defined as the number of evaluated workflow instances divided by the optimization latency. The *solver time* (or *solver lifespan*) is the time required for one solver execution: it includes the reading of all the input data of a job from the local disk, their processing and the writing of all the output data for the job on the local disk. Analogously *service time* (or *service lifespan*) is defined as the time required for one service execution. This time includes the downloading of input data, the solver latency, the uploading of output data.

A key element of the analysis performed in this thesis is the cost incurred by each organization participating in the workflow. The *service cost* is assumed proportional to the service time, whereas the *solver cost* is proportional to the solver time — a solver performs a license check-in as soon as it starts and the corresponding license checkout when it is about to finish. These definitions of the service cost and of the solver cost are associated with one workflow instance. The cost associated with the complete workflow is simply the sum of the cost associated with all the instances.

A *job* is a workflow instance with its related input data and input parameters. The workload is considered consisting of a specified number of jobs, called a *session*. The number of jobs concurrently being processed is referred as *multiprogramming level*. The *job latency* is the time required for completing a job, from its submission to the composite service to its termination. Finally, the *composite service time* is the sum of the service times for each service involved in the workflow.

The job latency is different from the composite service time because of the time spent along the links and because, in general, a schema could have some branches running in parallel. Note also that predicting in advance the relation between these indexes is hard because the transfer of data may overlap with computation and because the system may be concurrently executing several jobs.

Regarding the experiments presented in Chapter 6 all the latency and time results refer to the average values computed in a session.

This work takes also in account the point of view of the organizations participating in the composite service implementation. A key element of the analysis is the *cost* incurred by each organization. For each organization, the cost in processing a job is assumed proportional to the service time for that job, i.e., to the time actually spent by the job at that organization. In the experiments (described in Chapter 6) the cost globally incurred by all the organizations, i.e., the composite service time is considered.

The main assumption underlying this work is that the cost incurred by an organization for executing a job is proportional to the time spent by the job at that organization. This assumption is justified by the application domain of our interest, characterized by many large and resource-consuming jobs that are submitted in batches and do not require interaction with human users. Another important assumption is that the scheduling machinery should not rely on any dedicated hook from the composing services. So, for example, the composite service machinery cannot know whether one of the composing services is about to begin the execution of another heavy workload. While this assumption may limit the performance that can be obtained, we believe it is a practical way to facilitate the aggregation of organizations. Knowing that the composite service will properly balance performance and cost, moreover, may be a powerful factor for encouraging organizations to indeed share their services.

4.5 Communication Computation Overlap Pattern

The term *communication computation overlap pattern* is used to indicate the actions involved in the transfer of the output data produced by a source service to the target service that needs those input data. Services basically perform three major activities: *i)* download input data, *ii)* elaborate the data and *iii)* upload the output data. In current engineering workflows there is no overlap between these three activities, thus data elaboration does not start before the end of input data downloading. This pattern is very simple and does not leave room to performance optimizations. In this thesis are described and evaluated three more communication computation overlap patterns, currently these new patterns are not used in engineering services.

Working with engineering workflows we have identified four communication computation overlap pattern: NoPipe, InputPipe, FullPipe and OutputPipe.

4.5.1 NoPipe

NoPipe is the simplest communication computation overlap pattern, with this pattern there is no any overlap between communication and computation, this

is the pattern currently used by engineering services. Figure 4.3 shows the UML sequence diagram for the services with no overlap. When the service is invoked it starts the downloader and waits for the data arrival. At data arrival, the service logic starts the solver and wait for its termination. At solver termination, data is available for next services and the service invoke the uploader for the arriving requests. The license use depends only on the solver process.

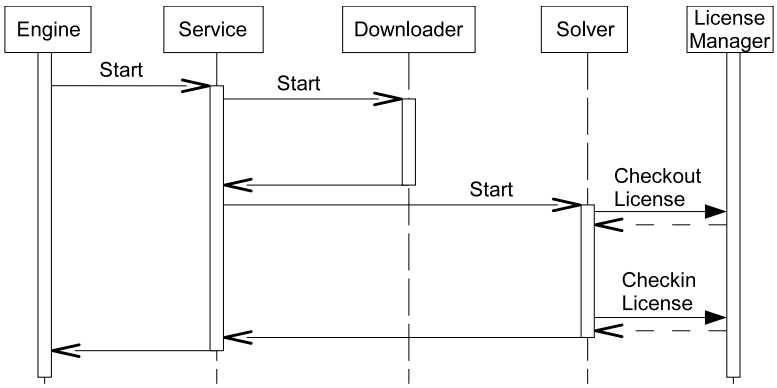

Figure 4.3: Sequence diagram for NoPipe.

With this pattern, the solver cost depends only on the local resources, e.g. it is not affected by links congestion or by the rate at which previous services are working. The service cost is roughly approximated by the sum of the time for downloading data, uploading data and the solver time. We expect this pattern to yield the best performances in terms of solver cost.

4.5.2 InputPipe

The InputPipe communication computation overlap pattern extends the NoPipe pattern introducing a pipeline between the dowloader and the solver, i.e. data elaboration is started as soon as the input data downloading starts. Figure 4.4 shows the UML sequence diagram for the services with the InputPipe communication computation overlap.

With this overlap pattern, the solver time is no more constant and it could be greatly affected, in a negative way, by the speed at which data is downloaded, in other terms the solver time depends on external factors such as links state or previous services upload speed. On the other hand while the downloader is still downloading data the solver is able to elaborate previous data chunks. Under the same external conditions the service time is expected to be lower than the NoPipe case. This pattern, should also positively affect the overall throughput.

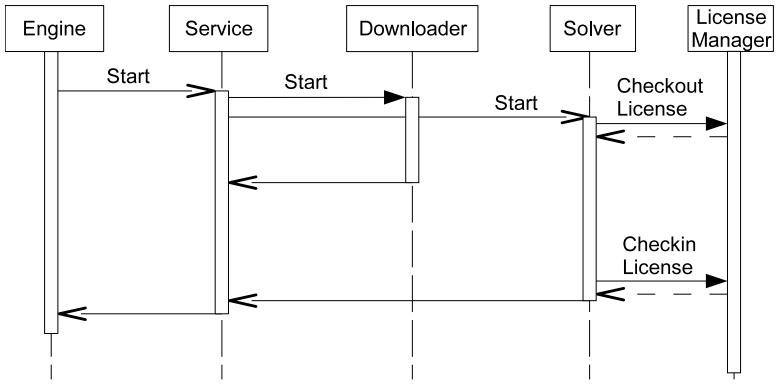

Figure 4.4: Sequence diagram for InputPipe.

4.5.3 FullPipe

The FullPipe communication computation overlap pattern is the most complex overlap pattern and it provides that downloading, elaboration and uploading start at service invocation. Figure 4.5 shows the UML sequence diagram for the services with the FullPipe communication computation overlap. The services functioning is pointed out showing two consequent services. As long as the previous services is ready to upload data, the workflow engine starts the next service, this service starts the downloader that in turn issues an upload request, the service starts the solver also, and issues and output start signal to the workflow engine. At this time the next service become a stage of the pipeline.

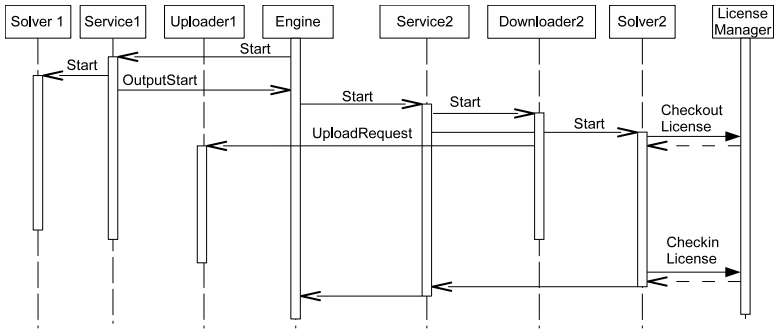

Figure 4.5: Sequence diagram for FullPipe.

With this overlap pattern, as for the InputPipe case, the solver time is no more constant, and it could be greatly affected, in a negative way, by external factors such as links congestion or previous services upload speed. On the other hand while the downloader is still downloading data the solver is able to elaborate previous data chunks and the uploader is able to upload such elaborated data. Under the same external conditions, thanks to elaboration and transmission overlap, we expect this pattern to provide better performance in terms of

overall throughput, but we also expect it to give worse results in terms of solver cost. The service time is expected to be lower than the NoPipe and the InputPipe case. This pattern, should also positively affect the overall throughput.

4.5.4 OutputPipe

Figure 4.6 show the UML sequence diagram for the services using the OutputPipe communication computation overlap. With this overlap degree the solver process starts when input data downloading is terminated.

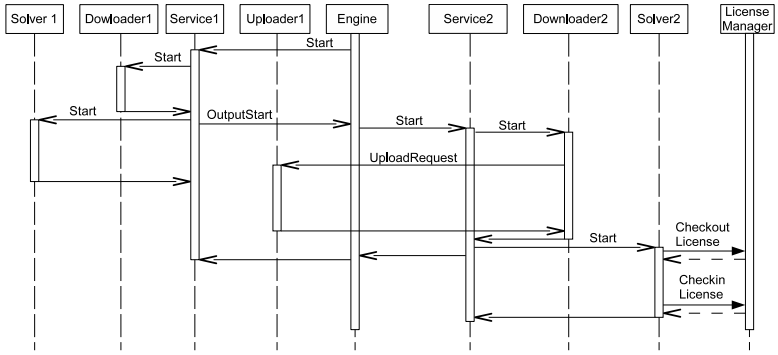

Figure 4.6: Sequence diagram for OutputPipe.

Similarly to the NoPipe pattern, because the solver works only on local data, the solver time is expected to be relatively constant.

4.5.5 Discussion

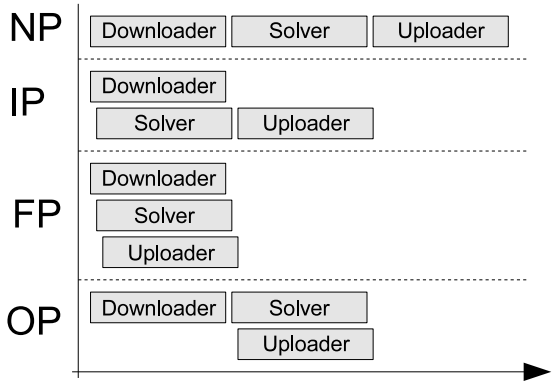

Figure 4.7: Communication computation overlap patterns.

Figure 4.7 grahically shows the overlap between the process involved in a service invocation. The choice of the computation communication overlap pattern is not always free. While the pipeline between the solver and the uploader is almost

always feasible in pratice, the feasibility of the pipeline between the downloader and the solver depends on the format of the input files and how the solver access the data contained in the file. A sequential access will be always compatible with the input pipeline while a random access will, in general, never be compatible with the input pipeline.

As will be showed in Chapter 5 and in Chapter 6, even in the case when the overlap pattern can be freely chosen, the trade-off between cost and speed must be considered, e.g. the FullPipe pattern allow (in general) a greater throughput, and a faster execution than NoPipe which is the pattern providing the lowest cost in terms of solver time.

4.5.6 Centralized and Distributed Engines

Engineering and scientific workflows must deal with data management [11], data management services have minimal impact on the execution of the workflow, they must satisfy stringent space and time constraints, and guarantee that no data is lost. Data management services must be first class services [41] or they can be included into other services, in both cases data management issues cannot be neglected.

In engineering workflows data is usually exchanged through a pull protocol, i.e., each service is responsible for downloading the data it needs. Of course, the workflow engine must notify services about the location of the respective input files.

In a distributed environment application, data associated to workflow processes need to "travel" along with the process. However, the application data should not be stored in the WfMS itself, for reason of efficiency, and hence a data manager is required. Specific data nodes can be needed for providing access to the data itself [5]. The activities manipulated by control node do not contain any data per se except a few simple variables (integer and real numbers, booleans, strings) used to evaluate control flow conditions. Part of these variables will be references (documents id's) to data stored in the data node. Thus, control nodes only need to transfer simple variables.

In the work of this thesis centralized and distributed data exchange between web services are evaluated [67]. In traditional service composition frameworks, the data-flow and the control-flow are centrally coordinated, and the composed service operates as the hub for all data communications. Infrastructures, supporting the service composition paradigm, employs a distributed data-flow approach that supports direct data exchanges among web services [44].

The distributed data-flows can avoid many performance bottlenecks attending centralized processing. Their prototype implementation demonstrates that distributed data-flow, combining with active mediation, is effective and more efficient than centralized processing when integrating large engineering software

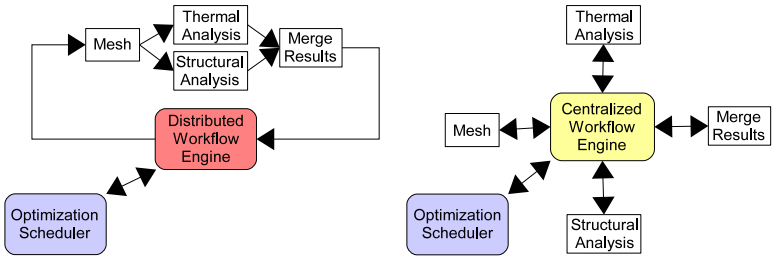

Figure 4.8: Centralized and distributed storage architectures.

services.

A design optimization system is usually implemented in a *centralized* way, as follows. There is a single repository that can be accessed by all modules (i.e., workflow engine, optimization scheduler, services). The workflow engine stores input and output data generated by the services in this repository. Each service reads input data from the repository, stores them on local storage and writes the final output data back to the repository.

When the services are distributed geographically, as in a multi-organizational system, it makes sense to explore a *distributed* storage architecture, in which the output data of a service are passed directly to the service that needs that data, i.e., without moving such intermediate results back and forth to the workflow engine. In this case, the repository at the workflow engine stores only the final results produced by each instance. The two alternatives are depicted in Figure 4.8.

4.6 Grid Workflows Challenges

Significant challenges arise from achieving performance objectives in large-scale, heterogeneous and highly dynamic Grid environments with shared computing and communication resources, and where the application behavior and performance is highly variable. There are many performance related parameters that must be dynamically tuned to match the Grid operating conditions and the application performance requirements.

As Grid applications grow in scale and complexity, and with many of these applications running in batch mode with limited or no direct runtime access to the application components maintaining the desired performance objectives using approach based on ad hoc manual tuning and heuristics is not just tedious and error prone, but unfeasible [11, 36, 70]. Services composed of other services or using Grid resources on the Internet must be largely self-managing, they must dynamically detect and respond, quickly and correctly, to changes in application behavior and state of the Grid.

Chapter 5

Multiprogramming Level Control

The number of concurrent workflow invocations, i.e. the number of concurrent design evaluations in design optimization terminology, is usually manually set by an expert operator. The *multiprogramming level* (MPL) is defined as the number of concurrent workflow invocations being executed by the workflow engine. The multiprogramming level has a strong impact in performances of design optimization system, and, in local systems, it is usually set considering the availability of computing and storage resources: software licenses, disk space, number of CPUs.

In the case of a static environment, the relation of the most interesting performance indexes with the multiprogramming level can be inferred by simple argumentations. The throughput is expected to raise quite linearly with the MPL for low MPL values, until it reaches a saturation level, after the saturation the throughput reaches a trashing area where it decreases because of the contention on shared resources. In general throughput it is considered a convex function of the MPL. On the other hand workflow latency is monotonic with MPL, composite service time is monotonic with the MPL too.

The composite solver time behaves differently with different patterns of communication computation overlap. For example when the communication computation overlap pattern does not affect the solver execution the composite solver time is expected to be almost constant, this is the case of the NoPipe (see Section 4.5.1) and OutputPipe (see Section 4.5.4) overlap pattern. With the other two pattern: InputPipe (see Section 4.5.2) and FullPipe (see Section 4.5.3) the solver execution depends on the input data, which in turn arrives with a rate bounded by the previous services and links throughput.

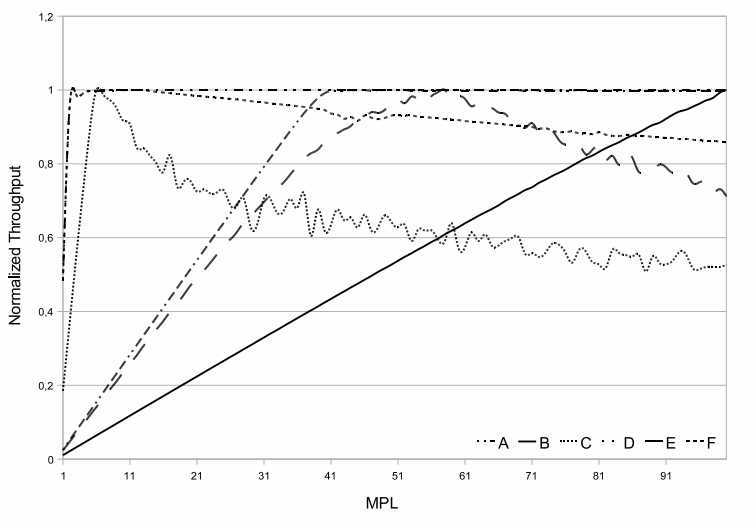

Figure 5.1: Throughput vs. MPL.

5.1 Fundamental Issues

It is important to understand the relation between throughput, composite service time and multiprogramming level. To explain the choices for the control of the multiprogramming level, this section anticipates some results presented in Chapter 6.

Figure 5.1 shows the throughput plotted against the multiprogramming level under several working conditions, working conditions differ from number of available licenses, link quality, size of exchanged data and communication computation overlap pattern. The considered multiprogramming levels are in the range from 1 to 100. The actual shape of the curve—including initial slope, saturation point, maximum throughput—greatly depends on a myriad of factors.

In brief, in case A throughput grows linearly with MPL and reach saturation for a average MPL. In case B, throughput never reach saturation and will increase far beyond the maximum MPL. In case C, throughput reaches saturation for low MPL values and then starts to decrease with a marked thrashing. In case D throughput suddenly reaches the saturation for very low MPLs, then it does not reach any thrashing condition. In case E throughput grows almost linearly with MPL, then it reaches the saturation area and then starts to decrease. Finally in case F, throughput grows very fast for low MPLs and then, after saturation, starts to decrease.

Figure 5.2 shows the composite service time plotted against the multiprogramming level under a variety of working conditions. Recall that composite service time has to be minimized to minimize the cost incurred by organizations participating in the workflow. In case A, the composite service time grows almost

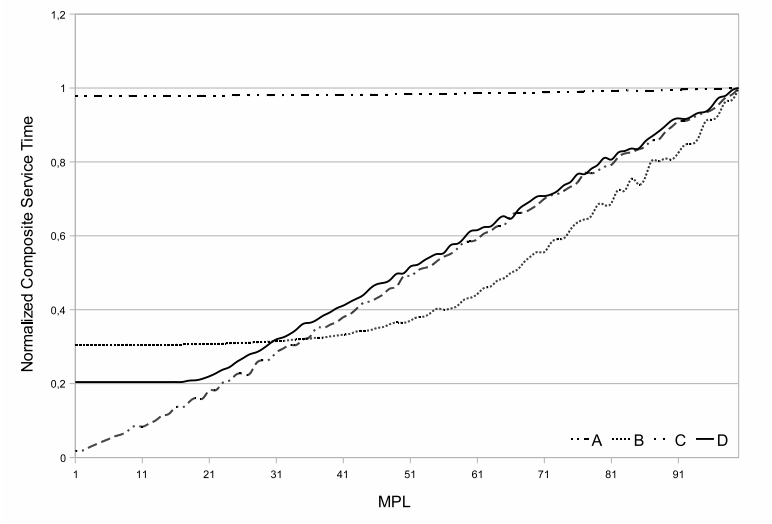

Figure 5.2: Composite service time vs. MPL.

linearly with the MPL, this suggests that in case A shared resources are always saturated. In case B composite service time is almost independent from the MPL for a wide range of MPL values, then it starts a slow growth, till reaching a linear growth. In case C the composite service time is almost constant in the considered multiprogramming level range. In case D the transition from constant composite service time to linear growth is much sharper than in case B.

It is clear from the above considerations that the choice of the multiprogramming level has a great impact on performances. The choice of the best MPL value is not univocally determined, even with only two goals such as maximizing throughput and minimizing cost we must deal with a trade-off. Use of a sufficiently high multiprogramming level, could be a simple yet effective way to maximize throughput, without having to figure out the shape of the curve in the specific, potentially unknown, environment*. Unfortunately, such a naive strategy would lead to high composite service time—thus to high cost globally incurred by the participating organizations. On the other hand the use of a very small multiprogramming level is the simplest way to minimize the cost but also the throughput.

We remark that, in practice, the actual shape of these curves is not known, because the values of the numerous relevant parameters is also unknown. Moreover, the values of these parameters may vary during execution, which obviously provoke a reshaping of the curves. It follows that determining a suitable value for the MPL in advance is not feasible.

*Depending on the system parameters, though, the system could exhibit a form of thrashing, i.e., throughput may collapse with a large MPL value. This point is irrelevant to our discussion, however.

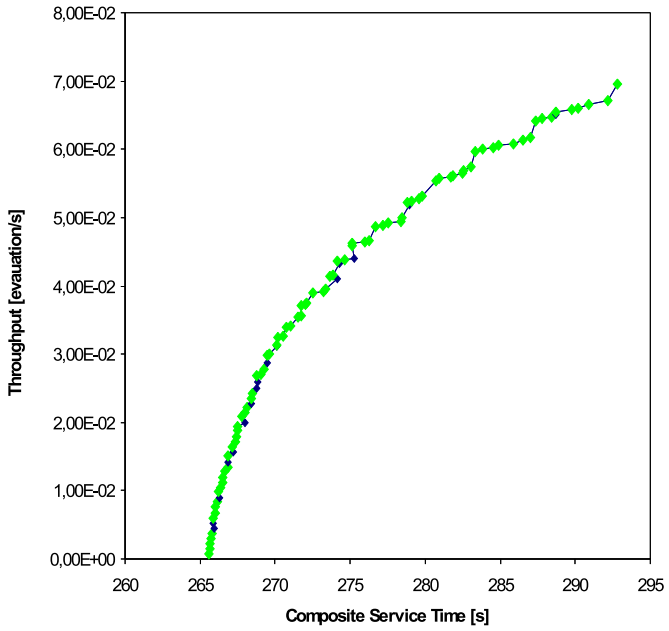

Figure 5.3: Throughput vs. composite service time with almost each multiprogramming level belonging to the pareto frontier (green dots).

The above considerations may be illustrated also by means of Figures 5.3, 5.4 and 5.5 shows three scatter plot under different scenarios. There is a dot for each multiprogramming level from 1 to 100. Optimal multiprogramming levels are represented by green dots. In particular Figure 5.3 shows a scenario where the considered MPL range does not reach the throughput saturation and thus, because increasing MPL leads to increased throughput, almost each MPL belongs to the pareto frontier. Figure 5.4 shows a similar situation, the only difference is that in this case throughput reaches saturation but it does not trash. In these two cases almost all MPL values (in the range between 1 and 100) are optimal. Finally in Figure 5.5 throughput produces a trashing behavior for large MPL values.

5.2 Overview of Our Proposal

Figuring out what are the performances in advance is very hard and nearly impossible, because of the myriad of factors involved that may influence the actual shape of the curves [59]. Moreover, these factors could even change dynamically and unpredictably due to the very nature of the Internet and of Internet-based services, where fluctuations in load and traffic are rather common events. In the approach presented in this thesis the multiprogramming level is varied dy-

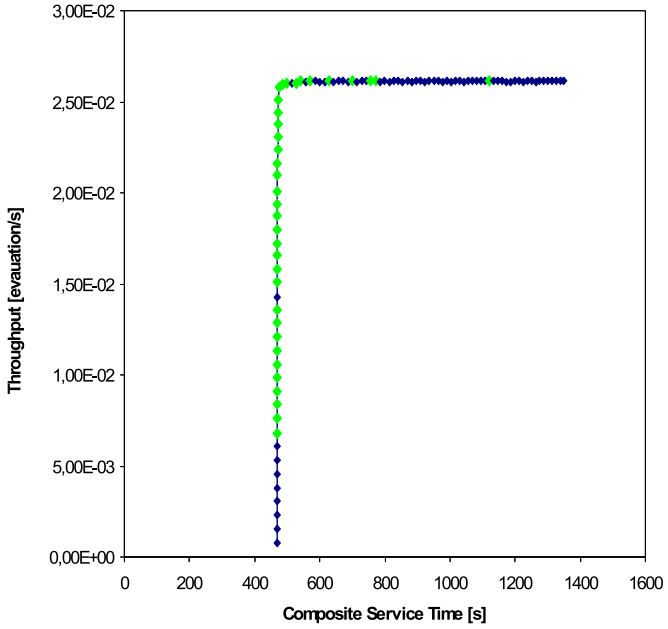

Figure 5.4: Throughput vs. composite service time with throughput exhibit saturation but with almost each multiprogramming level belonging to the pareto frontier (green dots).

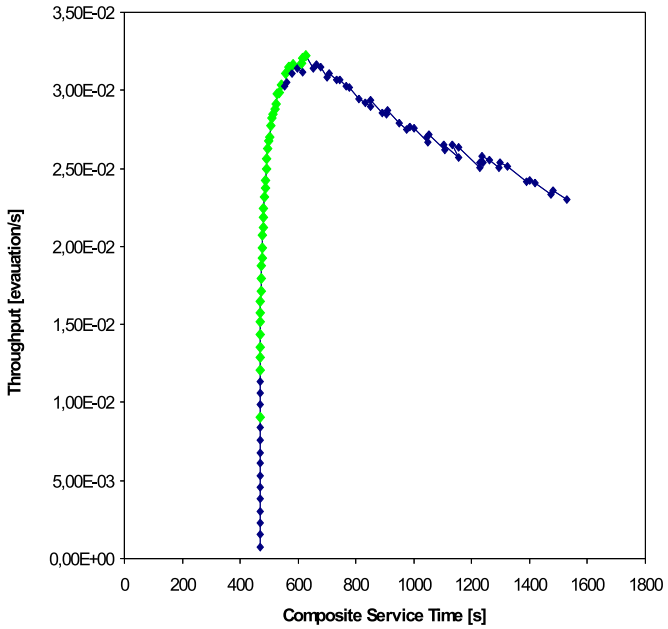

Figure 5.5: Throughput vs. composite service time with thrashing throughput.

namically and automatically to obtain a good trade-off between throughput and cost.

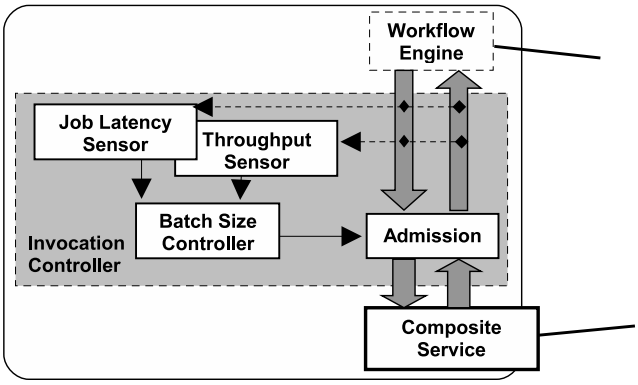

Figure 5.6: Invocation controller details.

The workflow engine is threated as a black box and an *invocation controller* is placed in front of it (Figure 5.6). Jobs submitted by clients to the workflow engine are intercepted by the invocation controller. The invocation controller selects dynamically a value for the multiprogramming level and implements a form of admission control to ensure that the number of in-progress jobs does not exceed that value. Excess jobs are queued by the invocation controller, that will inject them into the workflow engine as prior jobs complete. The current value for the MPL is provided by a *control algorithm* whose inputs are estimates of the current latency and throughput provided by dedicated *sensors*.

The key problems in this approach are how to enable the invocation controller to: *i)* select a suitable value for the MPL; and *ii)* vary that value dynamically if necessary. The basic idea underlying the solution is based on the following observations. The ratio $\frac{throughput}{job\ latency}$ (called *objective index*) usually varies with multiprogramming level according to a bell-shaped curve (Figure 5.7). The actual curve depends on many factors, but its shape is of the form in the figure.

This curve has two key features. First, its peak occurs at a multiprogramming value very close to the cost-optimal value. By varying the multiprogramming level dynamically so as to maximize the objective index, thus, the multiprogramming level becomes very close to its cost-optimal value. Second, the composite service machinery may measure the objective index on-line, by measuring the current values for throughput and job latency. Note that measuring the composite service time on-line is *not* possible, because there are no hooks from the composing services.

The control algorithm maximizes the objective index automatically and dynamically, thereby leading the multiprogramming level to a value close to the optimal one. This approach treats the system as a black box. The invocation controller does not need any prior or run-time knowledge about the static or

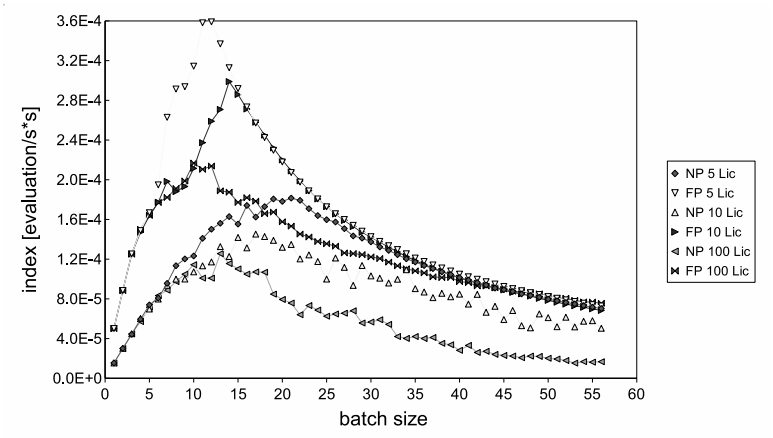

Figure 5.7: Objective index vs. multiprogramming level (batch size).

dynamic factors affecting performance: it simply observes, through its sensors, start time and completion time of each job. Note also the the composing services does not need to provide any hook to the invocation controller.

Essentially, each control algorithm starts with a low value for the multiprogramming level and increase this value until the measurements show that the objective index starts to decrease. At this point the multiprogramming level should be decreased (details in the next section.) This simple approach would not work by reasoning on either throughput or job latency taken in isolation. The curve throughput vs. MPL exhibits a flat or nearly-flat region beyond the saturation, thus maximizing throughput would lead to very wide oscillations of the MPL value. The job latency instead grows more or less steadily, thus minimizing this index would invariably keep the MPL to the minimum value.

5.3 Adaptive Invocation Controller

The controller algorithms work with the aid of two dedicated sensors. The latency sensor measures the job latency averaged across the last MPL of jobs that have completed. The throughput sensor measures the throughput across the same set of jobs.

The two sensors are notified by the invocation controller whenever a job is injected into the workflow engine and whenever a job completes (Figure 5.6). They use these events to build estimates for the current job latency and the current throughput dynamically. Such estimates, along with the corresponding estimate for the current objective index, are used by the MPL controller as described below.

The job latency estimate is an average of the observed job latency, evaluated over a number of observations equal to the current MPL. The throughput estimate

```
//the last objectiveIndex for which occurred a batch size change
double index initial 0;
//the sum of the relative differences between the last index an the current ones
double cumulatedRel initial 0;
//the current MPL, it cannot be lower than 1
int mpl initial 1;
//the MPL increment
int delta initial 1;

void main() {
   while (true) {
      sleep(getLatency());
      updateMPL(getObjectiveIndex());
   }
}

//newIndex is the actual estimated value for the desired index
void updateMPL(double newIndex) {
   double rel = (newIndex - index) / index;
   cumulatedRel += rel;
   if (rel < -0.1 || cumulatedRel < -0.2)  {
      delta = -delta;
      mpl = mpl + delta;
      index = newIndex;
      cumulatedRel = 0;
   } else if (rel > 0.1 || cumulatedRel > 0.2) {
      mpl = mpl + delta;
      index = newIndex;
      cumulatedRel = 0;
   }
   setMPL(mpl);
}
```

Figure 5.8: Timed controller psedo-code.

is also built over a number of observations equal to the current MPL. The details of the sensors include some additional complexity needed to cope with scenarios in which the number of available observations does not match the current MPL. In such cases a sensor may provide either the last computed estimate or a new estimate constructed optimistically. An optimistic estimate is constructed by taking optimistic values for the missing observations. If, for example, the number of available observations is k units below the current MPL, it is assumed that the next k jobs will complete immediately. Further details are omitted as irrelevant to the discussion.

We have experimented with four different policies for the adaptive invocation controller, called: TIMED, Derivative, TCP Like, and Parabola. We describe each of them in detail in the next subsections.

5.3.1 TIMED Controller

The controller runs periodically, with a period equal to the current job latency (as measured by the sensor). At each run it updates the *mpl* as follows. The update algorithm determines the current value for the objective index as provided by the sensors and determines the relative difference with the prior value (rel).

The algorithm also maintains a boolean state variable (*delta*) that is true when the MPL is growing and false otherwise. To follow the bell-shaped curve of the objective index, *mpl* is increased when *rel* is positive and *delta* is true, or when *rel* is negative and *delta* is false. Conversely, *mpl* is decreased in the two other cases. The amount of the variation applied to *mpl* is proportional to *rel* (10 times in our experiments) and bounded by 5. In order to filter out changes in the measured objective index that are too small to be truly significant, we actually modify *mpl* only when the absolute value of *rel* is greater than 10% of the current objective index value.

In order to accommodate scenarios in which the objective index changes very slowly with the MPL, we also accumulate the observed changes whenever we do not modify the MPL (*cumulatedRel*). In case the accumulated changes consistently denote a growth (or decrease) of the objective index, we update the MPL accordingly. In this case, the threshold that triggers the update has been set to 20% of the objective index value.

The working of the TIMED controller is shown in Figure 5.8. Note that *mpl* starts from the minimum possible value. The MPL controller executes the main() method in the figure, that updates the MPL with a period equal to the current job latency. The update algorithm takes as input the current value for the objective index as provided by the sensors (*newIndex*) and determines the relative difference with the prior value (*rel*). To follow the bell-shaped curve of the objective index, when *rel* is positive (i.e., $newIndex \geq index$) *mpl* is incremented, whereas when *rel* is negative *mpl* is decremented. All the variations are of the same fixed quantity, *delta*. In order to filter out changes in the observed objective index that are too small to be truly significant, we actually modify *mpl* only when the change is greater than 10% of the current objective index value.

5.3.2 Derivative Controller

Similarly to the Timed controller, the *Derivative Controller* is based on the difference of the objective index between the last two observations. The multiprogramming level is updating is based on the sign of the difference and on the relative difference. The controller runs periodically, with a period equal to the current job latency (as measured by the sensor).

Figure 5.9 shows the pseudo code describing the working of the derivative algorithm. At each run it updates the *mpl* as follows. The update algorithm determines the current value for the objective index as provided by the sensors and determines the difference with the prior value (*diff*). The algorithm also maintains a boolean state variable (*delta*) that is true when the MPL is growing and false otherwise. To follow the bell-shaped curve of the objective index, *mpl* is increased when *diff* is positive and *delta* is true, or when *diff* is negative and *delta* is false. Conversely, *mpl* is decreased in the two other cases. The amount

```
//accelerator
double accelerator initial 1;
double previousIndex initial 0;
int previousMPL initial 0;
int delta initial 1;

void main() {
   while(true) {
      sleep(getLatency());
      updateMPL(getObjectiveIndex());
   }
}

void updateMPL(double newIndex) {
   double diff = newIndex - previousIndex;
   double deriv = (newIndex - previousIndex) / (mpl - previousMPL);

   int newMPL = mpl;

   if (diff < 0) {
      delta = -delta;
      accelerator = delta;
      int abs = max(abs(accelerator), abs(deriv) * 10);
      newMPL = max(1, mpl + delta * abs);
      previousIndex = newIndex;
   } else if (diff > 0) {
      accelerator = 2 * accelerator;
      int abs = max(abs(accelerator), abs(deriv) * 10);
      newMPL = max(1, mpl + delta * abs);
      previousIndex = newIndex;
   } else if (newIndex > optIndex) {
      delta = -1;
      accelerator = delta;
      int abs = mpl / 2;
      newMPL = max(1, mpl + delta * abs);
      previousIndex = newIndex;
   }
   setMPL(mpl)
}
```

Figure 5.9: Derivative controller pseudo-code.

of the variation applied to *mpl* is proportional to the estimated derivative *deriv* (10 times in our experiments).

In order to follow steep and lasting slopes, the algorithm maintains the *accelerator* value, that is the minimum variation amount for the MPL, the accelerator value is doubled each time the delta maintains its value and it reset when delta change its value. To early recognize changes in the environment conditions, the algorithm compares the current objective index estimate with the optimistic estimate, whenever the current index is greater than the optimistic estimate the algorithm reset the accelerator and reduce the MPL to a safer value.

5.3.3 TCP Like Controller

The *TCP like controller* incorporates some ideas of the TCP congestion control algorithm. The controller includes a state variable called *threshold* which determines the amount of the increment. If the current MPL is below the threshold, the controller increments the MPL by one (which mimics the slow start phase of TCP congestion control). If the current MPL is above the threshold, the controller increments the MPL by $1/mpl$ (which mimics the congestion avoidance phase of TCP congestion control).

Whenever a job starts, the controller associates a timeout with the job, whenever the number of jobs not completing whitin the associated timeout exceeds 50% of the current bach size the controller halves the MPL and it sets the threshold to the current MPL. Initially, the MPL is 1 and the threshold is set to maximum MPL allowed by the admission controller (100 in our experiments). In early experiments we adopted an approach closer to that of TCP, i.e., one single timeout was enough to trigger a MPL decrease, but we found this approach to be excessively conservative. In other words, the MPL decreases when too many of the jobs that have been injected do not complete by their timeout.

The timeout choice is done as follows. This controller also works with a latency sensor and a throughput sensor. In this case, however, the former measures the latency of the last completed job whereas the latter provides a throughput estimate that is updated whenever a job completes. When a job J completes, the current throughput estimate becomes the MPL that was current when J started divided by the latency experienced by J.

This controller attempts to follow the bell-shaped curve of the objective index in a way quite different from the previous ones (TIMED and Derivative). The value of $objIndex$ at step i, denoted $objIndex_i$, is $\frac{mpl_i}{lat_i^2}$ (this follows immediately from the working of the sensors). If one requires that $objIndex_{i+1} > objIndex_i$, straightforward calculations lead to $mpl_{i+1} > objIndex_i \times lat_{i+1}^2$, thus to $lat_{i+1} < \sqrt{\frac{mpl}{objIndex_i}}$. The timeout for the i-th job is set to the calculated lat_{i+1}. The reason is because if the job has not completed by that deadline then the objective index will not increase. In order to increase the stability of the controller, the above formulas actually make use of a low-pass filtering of the objective index estimates.

5.3.4 Parabola Controller

The *Parabola controller* is based on the Parabola Approximation [34]. With this control algorithm the objective index is approximated with a polynomial of degree 2 of the multiprogramming level, $objIndex = \alpha_0 + \alpha_1 mpl + \alpha_2 mpl^2$. The coefficients α_i are calculated using a recursive least square estimator with exponentially fading memory [76].

The multiprogramming level is updated as follows:

- if $\alpha_2 >= 0$, the approximation has not the correct concavity, then $mpl_{i+1} = mpl_i - 1$;
- if $\alpha_1 <= 0$, the parabola vertex correspond to a negative value, then $mpl_{i+1} = mpl_1 - 1$;
- else $mpl_{i+1} = \frac{-\alpha_1}{2*\alpha_2}$

Chapter 6

Experiments and Results

In this work, performances of an inter-organizational design optimization system have been evaluated in a variety of scenarios. Predicting the relation between these indexes is hardly feasible, even in a system where relevant parameters never change during execution — e.g., the transfer of data may overlap with computation and the system may be concurrently executing several jobs.

6.1 Simulator Design and Implementation

This section describes the design and the implementation of the discrete event simulator, introduces the reasons behind the choice of the discrete event option and compares the simulator with other simulators developed by the scientific community. The final section introduces some technological issues regarding the coupling of the implemented simulator with the simgrid framework [17].

The discrete event simulator is based on the DESMO–J framework [24], a general purpose framework for rapid building of event driven simulation models in Java. In the simulator we have realized several components, these components belong to the following areas:

- computing resources;
- networking;
- admission control;
- workflow schema;
- perturbation;
- workflow execution.

6.1.1 Introduction

To evaluate the performance of distributed applications we considered several methods. Analytical models, such as Queuing Networks [42], are almost impossible to use to rank different algorithms or to compare different architectural choices. At a first sight the simplest strategy would be to perform actual experiments by implementing real applications on real resources. This is not feasible for at least three reasons, first real applications might run for long periods and thus is not viable to perform a statistically significant number of experiments. Second, using real applications make it difficult to explore a wide variety of configurations. Third the environment is not controllable and variations in resource availability or load over time make it difficult to obtain repeatable results. In distributed computing, simulation is the most useful approach to effectively evaluate and compare algorithms and other configuration choices [17].

6.1.2 Workflow Description

Workflows are described by a *workflow schema*. A workflow schema contains a list of activities, for each *activity* is defined a *dependency set*, the first activity of a workflow schema has no dependency. A dependency set specifies which data and from which activity must be supplied to the activity.

Two kinds of data have been identified: *transfer* file and *support file*, transfer files are data file produced by an activity and used by next activities, their location is relative to the activity that has generated them; on the other side support files are files that do not depend on any activity for creation, they already exists when the workflow engine enacts a workflow schema. Dependencies between activities define the *data flow*, the *process flow* is implicitly defined by the data flow.

6.1.3 Computing Resources

Computing resources are composed of licenses, CPUs, memory and disk space. This four quantities take in account four kind of resources: cost of software, time, volatile space, permanent space. The main entity is the *host*, a host is a *server* composed of a *file system* and a certain number of CPUs. The file system entity is composed of a disk, a buffer cache and a list of files. Each operation on the disk is mediated through the *buffer cache*, the buffer cache has a fixed size and buffers are replaced using a last recently used (RLU) policy.

The *disk* entity represents an unlimited logical disk characterized by: sector size, read time, write time, and a list of allocated sectors. The processes access local data via files, because files are part of the file system every read or write request is mediated by the file system and the buffer cache. File system metadata is not stored on the disk. Read and write operations are queued in the same fifo

queue. A *file* is modeled as an array of disk sectors, we have implemented two kind of file: regular and fifo. The host filesystem is shared among all the processes running on that host. Within our modeling any host can run any application.

License managers are the container for the licenses. Licenses are available on a per-activity base, so each activity specifies to which license manager is bounded. Several activities can be bounded to the same license manager and thus they will share the use of the available licenses.

For the sake of our study the number of disk sectors and the memory at each host are considered unlimited, furthermore the major costs in setting up a server reside in the number of CPUs and in the license acquisition. Because the cost associated to license acquisition can be considerer the major part of the total cost for running a server, we assume that in each host there are unlimited resources (memory, disk space and CPUs), or in other words there are enough resources to run solvers, services and accessory processes (downloader and uploader) in a smooth way.

6.1.4 Networking

The *network* is implemented as a graph in which vertex are represented by hosts and edges are represented by unidirectional links. Each link is characterized by two static parameters: *i)* capacity and *ii)* delay and it is dynamically characterized by three properties:*i)* the set of the open connections, *ii)* the set of data chunks entering the link and *iii)* the set of chunk travelling through the link. Delay represents the time spent by data chunks through the link while capacity represents the speed at which data enter the link. Multiple processes can send data through an open connection and they share the link capacity that is equally shared by the processes.

Processes create connections over a specified link in respond to a file transfer request, then they write and read data to and from the connection a chunk at time. A chunk is a unit of data with a size equal to the disk sector size.

6.1.5 Workflow Execution

The *workflow management system* (WfMS) is the main entity involved in workflows execution. The same WfMS can handle more requests and more workflow schema. A WfMS contains a list of known workflow schema with the associated admission control. The *admission control* is the module responsible for controlling the number of the instance or enactment of the same workflow schema for that WfMS. Each admission control module contains the logic for queuing the incoming evaluation requests and it contains a specific control algorithm. The control algorithm monitors the progress of evaluation invocation an adapt the multiprogramming level of the admission control in accord with the implemented

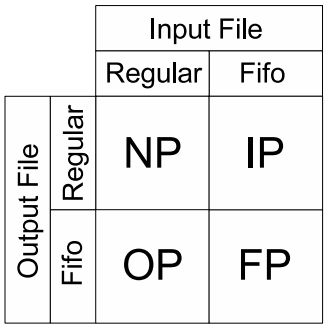

Figure 6.1: Input/Output files kind and communication computation overlap.

policy.

Clients require the WfMS to execute a specific workflow schema, the WfMS identifies the admission control for that workflow schema and forward an executable workflow instance to it. Then the admission control enqueues the workflow instance. When the number of workflow instances running is below the multiprogramming level for that admission control workflow instances are fetched from queue and they are started.

Workflow instance execution starts with the invocation of the service binded to the start activity. Then according with the workflow execution advance, workflow schema and the communication computation overlap pattern, the workflow engine invokes the next services. Services are wrapper to solvers, each service is responsible for downloading *input data* from previous services, invoking the solver and uploading *output data* toward next services. Solvers represent the specific software to invoke to transform input data into output data. Depending on the communication computation overlap pattern solvers read and write to regular files (in absence of pipelining) or to fifo files (when using pipelines), see Figure 6.1.

Solvers are characterized by three static parameters: startup time and elaboration speed. Startup time is the time required to start the solver and elaboration speed is the speed at which the solver elaborates input data. Furthermore for each solver output file is specified the size ratio when compared with the solver input file. The solver works as follows, the solver is invoked by the service wrapper, it waits for the specified startup time, then till there is input data reads a chunk of data from the input file, checkouts a license, elaborates the input data at the specified elaboration speed, checks-in the license to the license manager, and then writes the elaborated data to the output file.

To simulate variations in available resources we have introduced some processes with the aim of perturbing the available resources. The optimization process is modeled as a client of a specific WfMS. This process require workflows evaluation in order with its submission policy as described in Section 3.2.

The model can be defined in two ways: *i)* via a Java API or *ii)* via an XML description. The Java API lets us to program the simulator using the Java language, while the XML definition let us to define a model without the need to compile the definition itself and it simplifies the experiments parameterization.

6.1.6 Related Works

Simgrid is a simulation toolkit for the study of scheduling algorithms for distributed applications [17]. This toolkit is oriented toward the optimization of the assignment of a set of tasks onto a set of resources. They define a schedule as the assignment (in both time and space dimensions) of task to resources. Simgrid is written in the C language, and it is based on an event-driven simulation.

There are three major components in the Simgrid models: *i)* resources, *ii)* topology, and *iii)* tasks. Resources are modeled by two performance characteristics: latency that is the time needed to access the resource and service rate that is the speed at which the resource elaborates work units. Topology defines the interconnection between resources. In Simgrid there are two types of tasks: computing and transfer tasks. Computing task are CPU bounded while transfer tasks are bandwidth bounded, aside this distinction the simulator does not differentiate between computing and transfer task. At this time, the Simgrid toolkit does not provide any facility to simulate file-systems and disk.

The whole Simgrid toolkit is composed a several layers, to take advantage of the resource and the topology management our simulator have been interfaced with the MSG layers. While the different implementation technologies, Java and C, did not prove to be an obstacle to the integration, the very different nature of the two event-driven simulator engine threading model revealed that the integration could deal with only a very limited number of concurrent processes.

GridSim is a Grid simulation toolkit for resource modeling and application scheduling for parallel and distributed computing [33]. This Java-based toolkit is aimed at the investigation of effective resource allocation techniques based on computational economy over large scale grids. Simgrid provides a very limited network implementation that is not able to describe all the aspects considered in this thesis.

6.2 Static Experiments

This scenario and suite of experiments have been included in [67]. We consider a workload consisting of a specified number of jobs. We conducted experiments on a simple workflow schema, composed of four identical services serially connected as shown in Figure 6.2. In each scenario, we study the behavior of a *block optimization scheduler*, i.e., one injecting a batch of candidate designs when the previous batch has been completed, and that of a *steady* optimization scheduler,

i.e., one injecting a new design as soon as evaluation of the previous one has completed.

Moreover, we study all these scenarios under two different patterns for communication computation overlap, one in which a service starts only when all input data have been downloaded (NoPipe) and one in which a service starts as soon as the first chunk of input data is available (FullPipe). Our analysis provides useful insights into the numerous trade-offs involved in the implementation of such a complex scenario. We assume that at each stage the size of the output data is equal to the product of the size of input data for the output/input ratio, each service behaves as described in Section 4.3.

We performed a number of experiments around a working point described by the parameter values in Table 6.1. The buffer cache size is expressed in number of disk sectors. We modeled the disk access times as random variables with uniform distribution. We modeled link delay and bandwidth as random variables with normal distribution, with standard deviation equal to 1/10 of the mean value.

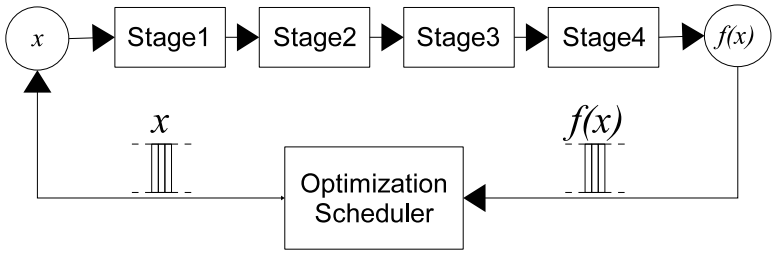

Figure 6.2: Serial workflow schema.

Table 6.1: Working Point

Parameter	value
Buffer Cache Size	100000
Minimum Disk Read Time [s]	0.001
Maximum Disk Read Time [s]	0.002
Minimum Disk Write Time [s]	0.0002
Maximum Disk Write Time [s]	0.0004
Sector Size [B]	8192
Initial Data Size [B]	1000000
Multiprogramming Level	10
StartUp Time [s]	30
Elaboration Speed [B/s]	65536
Output/Input Ratio	1.1
Mean Link Delay [s]	0.001
Number of Batches	20

All the experiments have been conducted with both storage architectures

(centralized vs. distributed), with both scheduler policies (block vs. steady) and with two computation communication overlap pattern: NoPipe and FullPipe. This analysis has enabled us to gain insights into the relationships between the numerous systems available. We remark, however, that the modules used in contemporary design optimization systems may restrict such choices. In particular, some optimization algorithms are inherently steady while others are inherently block based. Moreover, the majority of solvers do not support the FullPipe communication computation overlap pattern.

The following three subsections show the obtained results. The first is related to the NoPipe pattern, the second is related to the FullPipe pattern and the third describes the results in an aggregated way. Each plotted value on the charts is the mean of 20 simulation runs.

6.2.1 Results for NoPipe

Figure 6.3 shows throughput as a function of link bandwidth (note the logarithmic scale on the Y-axis). For small network bandwidth values, bandwidth is the system bottleneck then for larger values system throughput becomes independent from the bandwidth, the storage architecture and the scheduler policy. The steady policy achieves greater throughput than the block policy, the distributed storage achieves greater throughput than the centralized one.

It can be seen that when the link bandwidth is the system bottleneck, a Steady scheduler achieves better throughput than a Block scheduler. The improvement is about 200% for bandwidth in the range 10000-100000 B/sec, which may be representative of the bandwidth available in a WAN environment. As expected, when the bandwidth is no longer an issue, throughput is limited by other bottlenecks and becomes independent of both the storage architecture and scheduler policy.

Figure 6.4 describes the effect of the storage architecture. It shows the ratio of throughput with distributed storage to throughput with centralized storage. It can be seen that a distributed storage is always beneficial for a block scheduler, whereas for a steady scheduler the difference in the two cases is less sensible. Obviously, when the link bandwidth is no longer a bottleneck, whether the architecture is distributed or centralized becomes irrelevant from the point of view of throughput (ratio tends to 1).

Figure 6.5 shows the service lifespan measured on the fourth stage of the workflow schema versus the link bandwidth. It can be seen that the centralized storage guarantees a lower lifespan (note that the Y-axis has a logarithmic scale). The improvement over the distributed storage is about 100% for bandwidth in the range 10000-100000 B/sec, which may be representative of the bandwidth available in a WAN environment.

Figure 6.6 shows the effect of the storage architecture, in terms of ratio of

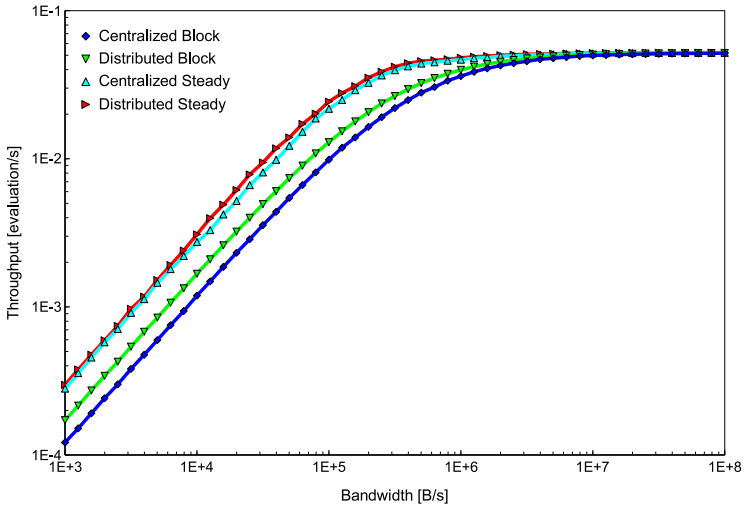

Figure 6.3: NoPipe throughput.

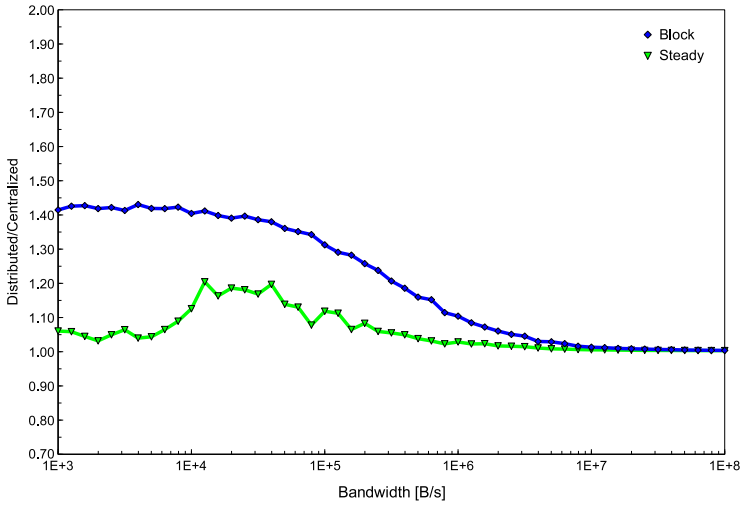

Figure 6.4: NoPipe throughput ratio (Distributed/Centralized).

service lifespan in the distributed case vs. service lifespan in the centralized case. Once again, distribution is more beneficial for a block scheduler especially when the available bandwidth is small. The solver latency is not shown because with the NoPipe pattern it is essentially independent of the link bandwidth.

6.2.2 Results for FullPipe

Figure 6.7 shows the throughput versus the link bandwidth, almost all curves are superimposed (on a logarithmic scale) within an envelope less than 10%. Figure 6.8 shows the throughput speedup and it confirms that the maximum

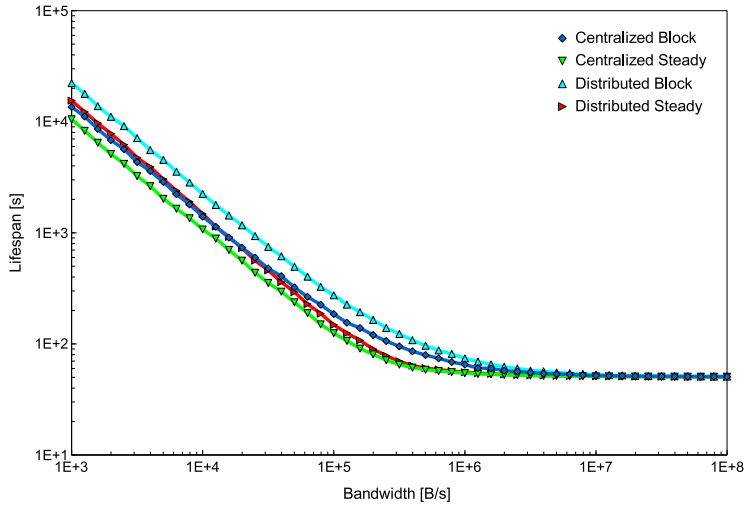

Figure 6.5: NoPipe service lifespan.

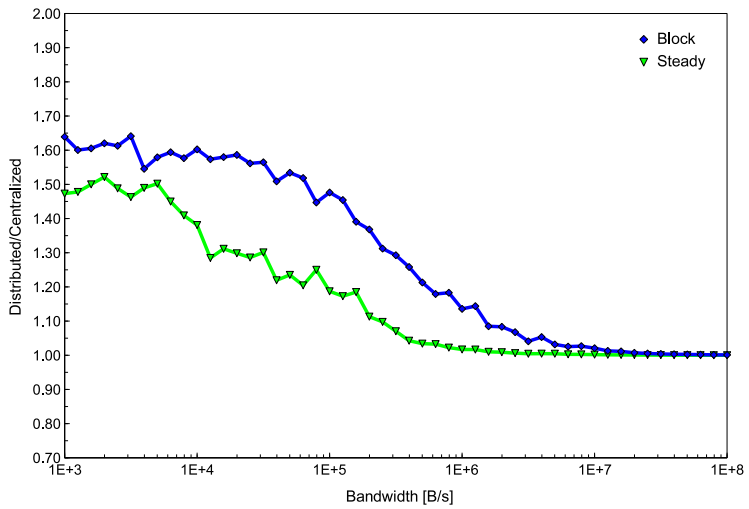

Figure 6.6: NoPipe service lifespan ratio (Distributed/Centralized).

throughput difference between centralized and distributed storage is around 10%. Figure 6.9 shows the service lifespan versus the link bandwidth, Figure 6.10 shows the latency ratio of the service lifespan measured with the distributed and the centralized storage. With the considered bandwidth range and with the FullPipe pattern, throughput and service lifespan are practically unaffected by the scheduler and the storage architecture.

Figure 6.11 shows the solver latency versus the link bandwidth. The solver latency with the block scheduler is greater than the solver latency with the steady scheduler, and the solver latency with the distributed storage is greater than

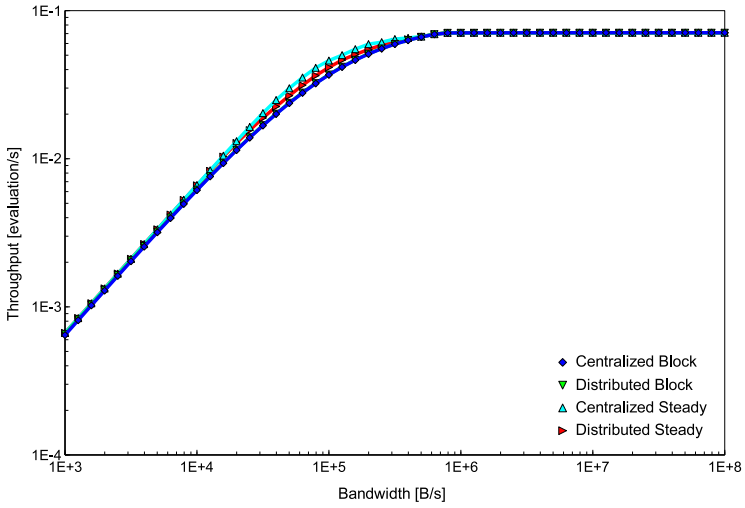

Figure 6.7: FullPipe throughput.

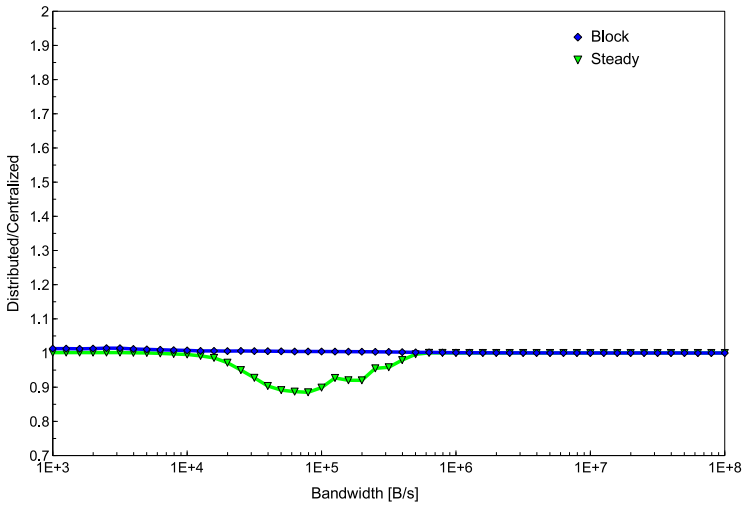

Figure 6.8: FullPipe throughput ratio (Distributed/Centralized).

solver latency with the centralized storage. Note that Figure 6.11 also includes the solver latency for the NoPipe pattern (this value is constant for the entire bandwidth range).

Figure 6.12 shows the solver latency ratio of the solver latency measured with the distributed and the centralized storage. For low bandwidth values distributed storage with the steady scheduler experiences a slowdown ranging from about 20% to about 90%.

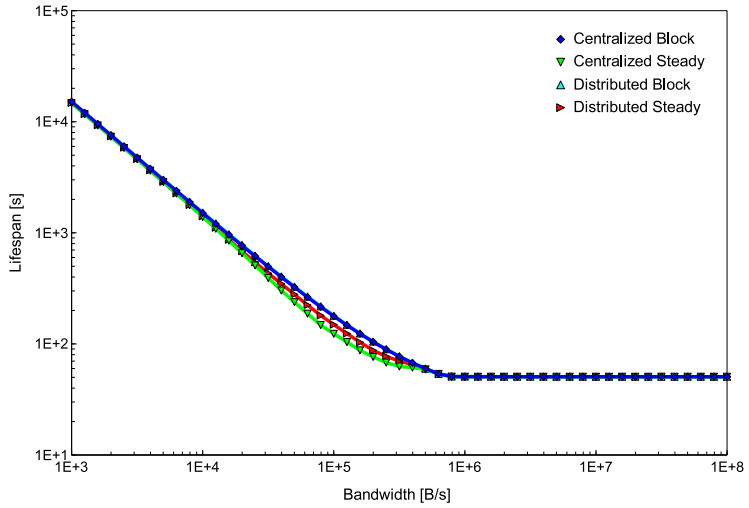

Figure 6.9: FullPipe service lifespan.

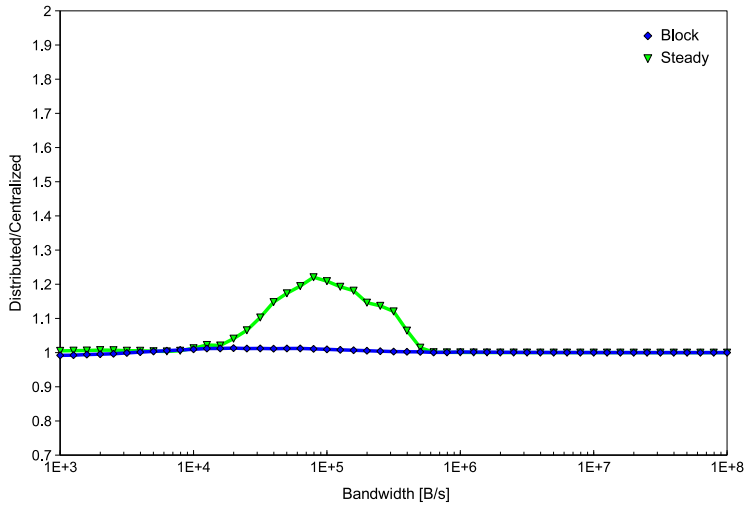

Figure 6.10: FullPipe service lifespan ratio (Distributed/Centralized).

6.2.3 Aggregated Results

Figure 6.13 shows the ratio of throughput with the FullPipe pattern to the NoPipe pattern. This Figure says to us that, as long as the bandwidth is a bottleneck, we could have a significant throughput improvement when connecting the services in a data pipeline. On the other hand Figure 6.14 shows that the communication computation pattern has not a great impact on the service lifespan, the ratio of FullPipe to NoPipe pattern is never greater than 1.5 and in certain conditions it is lower than 1. Under such condition the FullPipe pattern is preferable also from the point of view of the service cost.

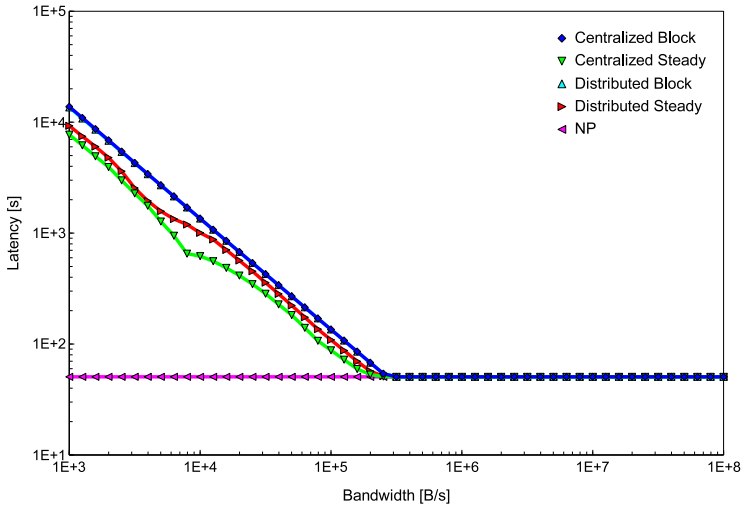

Figure 6.11: FullPipe solver latency.

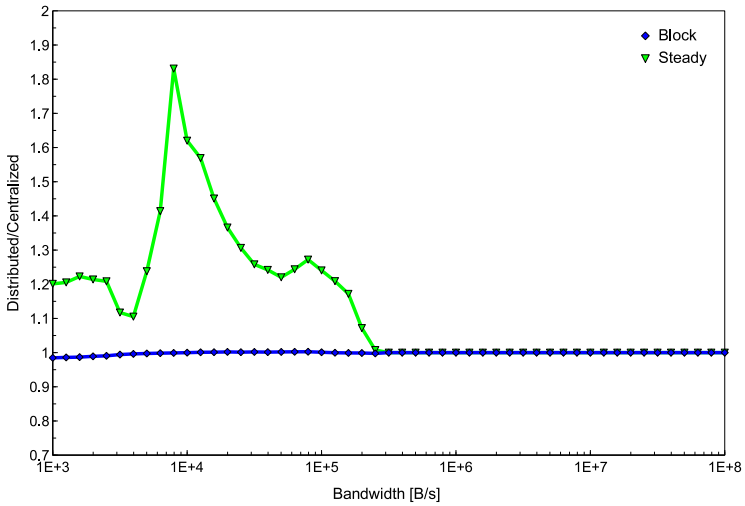

Figure 6.12: FullPipe solver latency ratio (Distributed/Centralized).

6.2.4 Discussion

A first significant finding is that when link bandwidth is an issue storage architecture, scheduler policy, communication computation overlap pattern may all have a significant impact on performance. Although this result is not completely unexpected, it was not obvious whether the performance implications of the various implementation options are indeed significant. In particular, we found it difficult to predict the effect of the interactions between them. According to our analysis, on the other hand, it seems evident that when one moves to a bandwidth-limited environment, the previously mentioned options do have a strong impact on per-

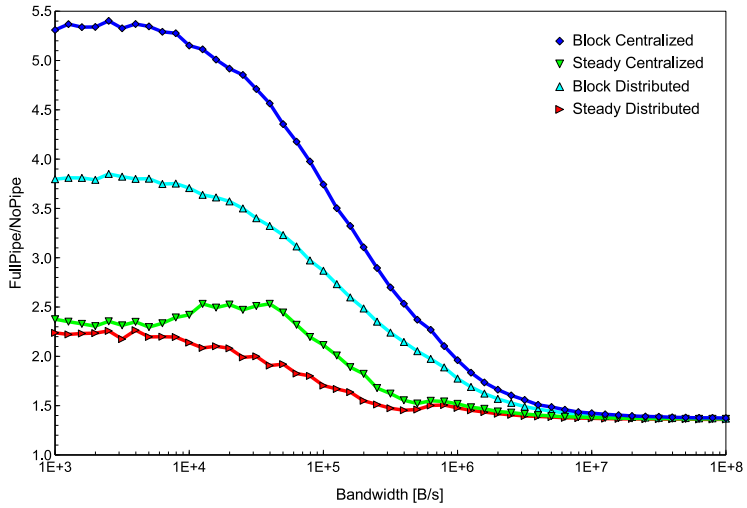

Figure 6.13: Throughput ratio (FullPipe/NoPipe).

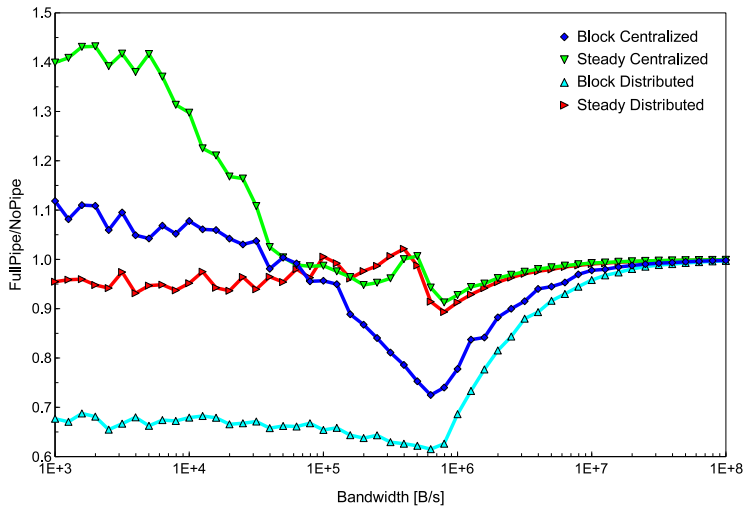

Figure 6.14: Service lifespan ratio (FullPipe/NoPipe).

formance and thus they have to be analyzed carefully.

Said this, it would be useful to provide useful rules for selecting the implementation options most suitable for a given scenario. It is evident, though, that these results cannot be used for predicting exactly the performance of any possible scenario, because of the large numbers of parameters involved and, perhaps most importantly, because the behavior of the overall system may be largely dependent on the specific workflow schema used.

Having warned that these results cannot be generalized easily, the key advices that we can be drawn from these simulations follow. Recall from the introduction

that throughput is the key performance index from the point of view of the user of the design optimization system, whereas service cost and solver cost are the indexes most relevant for the organizations that cooperate in the implementation of the system.

- When $\frac{bandwidth}{elaborationspeed} > mpl$, it basically does not matter how the system is implemented: all the options that we have analyzed exhibit more or less the same performance. When $\frac{bandwidth}{elaborationspeed} < mpl$, on the other hand, different options may lead to strong differences in performance.

- To maximize system throughput one should use a FullPipe overlap pattern. From this point of view, this strategy is always to be preferred to NoPipe even when bandwidth is not an issue (Figure 6.13). If FullPipe cannot be used (recall that many existing solvers do not support this option), then use of a steady scheduler is to be preferred over use of a block scheduler (Figure 6.3). In this case, whether storage is centralized or distributed is not very significant although distributed storage could provide some advantage.

- To minimize service cost one should use a centralized storage (Figure 6.5 and Figure 6.9). This conclusion is particularly useful, because a centralized storage is far simpler to develop, deploy and manage than a distributed one. A steady scheduler is to be preferred over a continuous one (Figure 6.6 and Figure 6.10). The choice between NoPipe and FullPipe is not univocally determined (Figure 6.14).

- To minimize solver cost one should use a NoPipe overlap pattern. We note, also in this case, that NoPipe is much simpler to implement than FullPipe and, in particular, directly supported by most existing solvers. Alternatively we can obtain some minor improvements using a steady scheduler (Figure 6.11).

- The steady scheduler is almost always preferable to the block scheduler. Many generation based optimization schedulers have variations to run in a steady–state way [66].

6.3 Controller with Static Environment

This scenario and suite of experiments have been partially included in [69]. This section describe experiments and result for evaluating the behavior of two controllers in a static environment. First of all performances have been evaluates, in a variety of different scenarios, as a function of a constant multiprogramming level (by keeping the invocation controller disabled.) Finally, the control algorithms performances have been compared with the multiprogramming level leading to the maximum throughput.

Table 6.2: Link parameters

Link Speed	Bandwidth Mean [KB/s]	Bandwidth StdDev	Latency Mean [ms]	Latency StdDev
High	977	97.7	0.1	0.01
Normal	97.7	9.77	1	0.1

Table 6.3: Working point

Parameter	Value
Buffer Cache Size	100000
Minimum Disk Read Time [s]	0.001
Maximum Disk Read Time [s]	0.002
Minimum Disk Write Time [s]	0.0002
Maximum Disk Write Time [s]	0.0004
Sector Size [B]	8192
Initial Data Size [B]	1000000
Start Up Time [s]	30
Elaboration Speed [B/s]	65536
Output/Input Ratio	1.1

We considered a workflow schema composed of four identical services connected as a pipeline (Figure 6.2). Each experiment is characterized by the following parameters: *i)* communication computation overlap pattern, either NoPipe or FullPipe Section 4.5; *ii)* number of licenses, either 1, 10, 20 or 100; *iii)* link quality, either Normal, High (Table 6.2.) The remaining parameters are summarized in Table 6.3, buffer cache size is expressed in number of blocks. Each experiment refers to a session composed of 10000 jobs. It is assumed that at any time, there is always at least one job submitted at the composite service and waiting to be executed (except when the session is about to complete).

6.3.1 Results for Constant Multiprogramming Level

The throughput results with constant multiprogramming level and invocation controller disabled are summarized in Figure 6.15 and 6.16, for the Normal and High link type, respectively. Each graph contains 8 curves, one for each combination of the pair communication computation overlap (NoPipe, FullPipe) and number of licenses (1, 10, 20, 100). Recall that throughput is measured across all jobs composing a session, thus the time for completing a session is the inverse of the throughput.

By comparing the values for the maximum throughput in the two figures

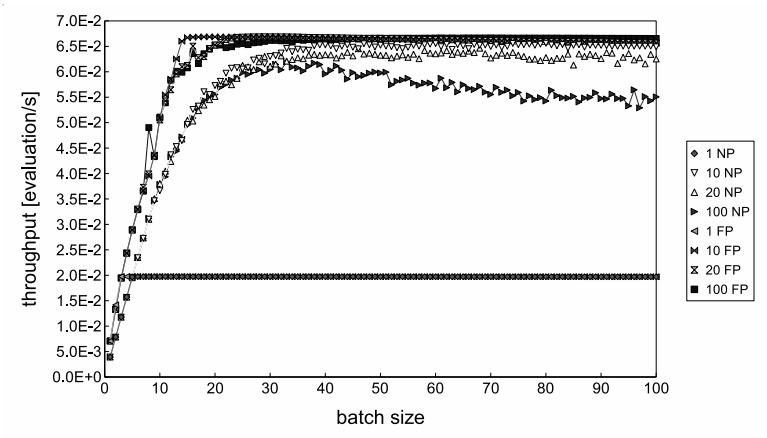

Figure 6.15: Throughput for constant multiprogramming level (batch size) with normal quality links.

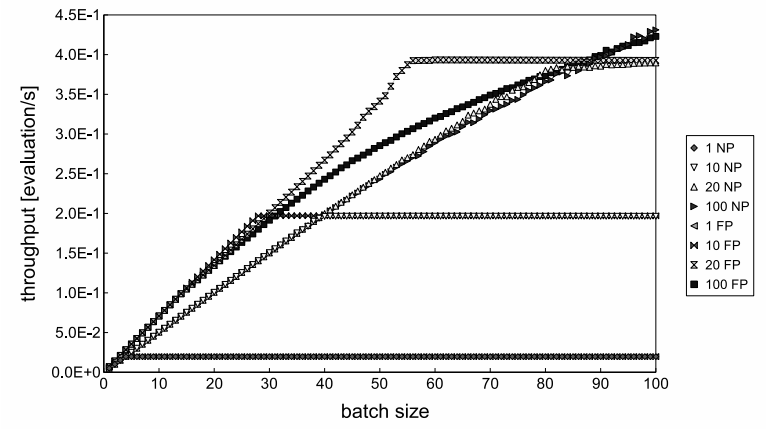

Figure 6.16: Throughput for constant multiprogramming level (batch size) with high quality links.

(note the different scale on the Y-axis) we deduce that with the Normal link type throughput is ultimately bounded by the capacity of the links. With the High link type, instead, the bottleneck is the number of licenses. We also note that: *i)* with the High link type, the saturation point for 100 licenses is reached for multiprogramming levels beyond those analyzed in our experiments; *ii)* the FullPipe pattern generally reaches the maximum throughput with a multiprogramming level smaller than the NoPipe pattern; *iii)* with the Normal link type and a large number of licenses, increasing the multiprogramming level may (very slightly) result in a form of thrashing.

The most important observation, though, is this: the minimum multiprogramming level leading to saturation depends significantly on the communication computation overlap pattern, the number of licenses, the link properties.

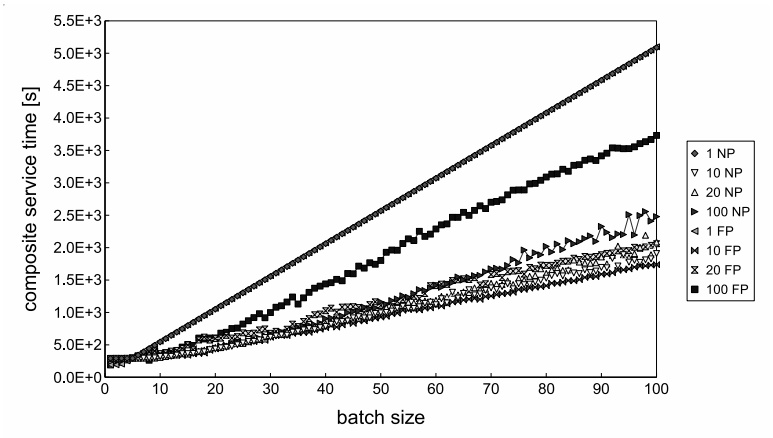

Figure 6.17: Composite service time for constant multiprogramming level (batch size) with normal quality links.

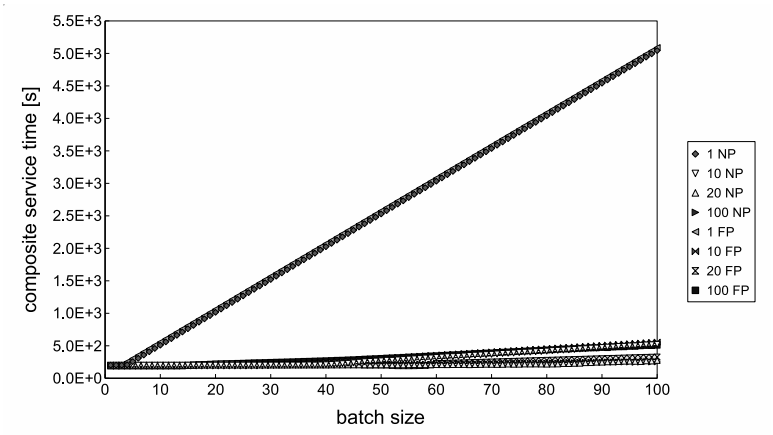

Figure 6.18: Composite service time for constant multiprogramming level (batch size) with high quality links.

We made similar experiments by varying the size of a job (in terms of input and output data) and obtained qualitatively identical results. Indeed, the effects of a larger job are very similar to those of a slower link.

The composite service time results with constant multiprogramming level and invocation controller disabled are summarized in Figure 6.17 and 6.18, for normal and high quality links, respectively. Each graph contains eight curves, with the same notation as above. The composite service time grows linearly with the multiprogramming level, the slope being dependent on the actual system parameters. The worst results (i.e., highest slope) are consistently those with 1 license.

These results confirm and corroborate the points made in Section 5.1. The value for the multiprogramming level must be a trade-off between two conflict-

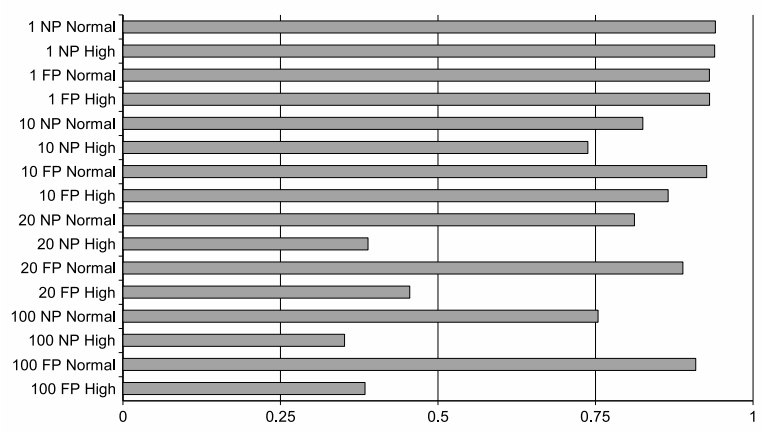

Figure 6.19: Throughput ratio (adaptive/best constant.)

ing needs, namely maximizing throughput while minimizing the composite service time (which determines the overall cost at the participating organizations). Unfortunately, selecting a suitable value is very hard. The minimal multiprogramming level leading to maximum throughput depends on many factors in a way that is hardly predictable; and, some of these factors may be unknown or changing in a hardly predictable way (e.g., link properties).

6.3.2 Results for Adaptive Multiprogramming Level

In this section the results obtained with invocation controller enabled are presented. To simplify their understanding we present them by using those found in the previous section as baseline.

Figure 6.19 shows the throughput results. There is one row for each scenario analyzed in the previous section. The value in each row is the ratio between *i)* throughput obtained with invocation controller enabled and *ii)* the maximum throughput that could be achieved with invocation controller disabled and constant multiprogramming level. For example, for the scenario with 10 licenses, NoPipe and normal quality links, the maximum throughput is *6.56E-2* while with the invocation controller the throughput reached *5.41E-2*, thus the ratio is *0.83*.

It can be seen that with 1 and 10 licenses the invocation controller reached a throughput quite close to the maximum attainable throughput (that, we remark once again, may be obtained with a multiprogramming level value that could not be known in advance). The worst throughput is that of the scenario "NoPipe, 10 licenses, High link type", slightly below 75% of the optimal value. With 20 and 100 licenses, similar remarks can be made for all the scenarios with a normal link type.

With a High link type and more than 20 licenses, instead, the loss in through-

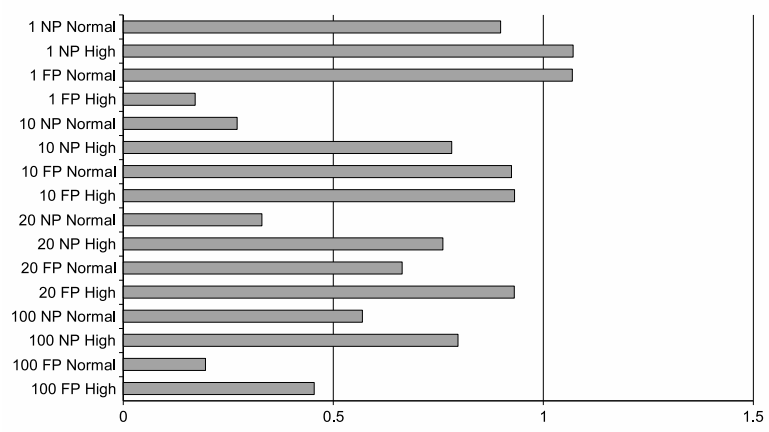

Figure 6.20: Composite service time ratio (adaptive/best constant.)

put is more substantial: less than 50% of the maximum throughput. The reason for this this unsatisfactory behavior is that the control algorithm increases the multiprogramming level in a very conservative way. When the optimal value is very high (recall the shape of Figure 6.16 with 20 and 100 licenses) reaching a good multiprogramming level value may take a long time.

Figure 6.20 shows the composite service time results in the same scenarios as above. Timed controller performances have been evaluated by using the same operating condition as in the previous table. That is, the baseline is the time corresponding to the minimum multiprogramming level leading to the maximum throughput (i.e., the throughput assumed as baseline in the previous table). The value in each row is the ratio between *i)* composite service time with invocation controller enabled and *ii)* baseline composite service time. For example, for the scenario with 10 license, NoPipe and a normal quality link, the composite service time related to the maximum throughput is *1530s* while using the invocation controller the composite service time is *414s*, thus the ratio is *0.27*.

In other words, the figure shows the reduction in cost (across all the organizations) when using the adaptive invocation controller. This is the most meaningful comparison, as one could minimize the cost (composite service time) by simply using a very small multiprogramming level, e.g., by injecting only one job at a time. Of course, in that case throughput would collapse.

Consistent reduction of the cost is observed in nearly all cases. The only exception being two of the scenarios with 1 license. In these two cases, however, we may argue that the increase is very moderate and could be more than compensated for by the ability of finding a "sufficiently good" multiprogramming level value in a fully automatic and adaptive way. We plan to address also this issue, however.

To summarize, the mechanism is indeed capable of trading off a small loss

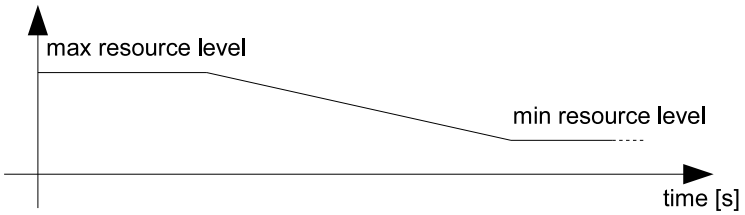

Figure 6.21: Resource availability level with linear decrease perturbation.

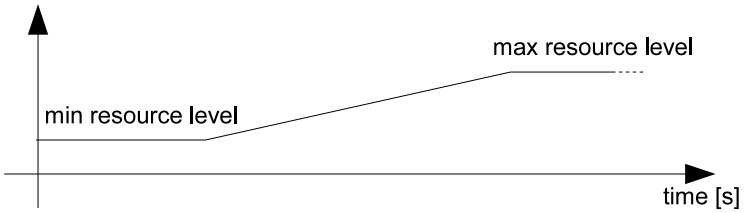

Figure 6.22: Resource availability level with linear increase perturbation.

in throughput against a decrease in the cost at the participating organizations. From the point of view of clients, completing a session would thus take slightly longer. From the point of view of the participating organizations, however, executing a session would consume less resources. Each organization would thus need less resources for executing a given amount of work, thereby enabling the accommodation of further additional workloads, either within the organization or submitted from the outside.

6.4 Perturbations

Real world conditions are never static, in this thesis, to mime perturbations on resource availability, four types of perturbation on the working conditions are considered:

- linear decrease,
- linear increase,
- correlated decrease,
- correlated increase.

With the linear decrease perturbation (see Figure 6.21) the level of a resource is decreased with a fixed periodicity, similarly with linear increase (see Figure 6.22) the level of a resource is increased with a fixed periodicity. Instead with the correlated decrease (see Figure 6.23) the level of the resource is decreased

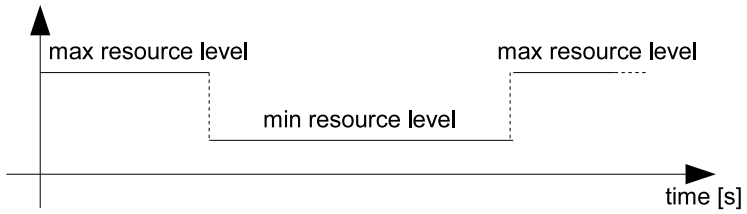

Figure 6.23: Resource availability level with correlated decrease perturbation.

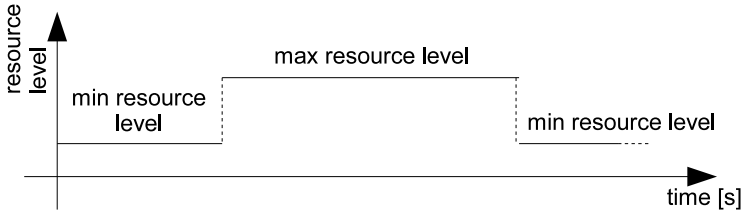

Figure 6.24: Resource availability level with correlated increase perturbation.

for a time dependent on the optimization process advancing, similarly with the the correlated increase (see Figure 6.24) the level of the resource is increased for a time dependent on the optimization process advancing.

To have a better evaluation of the control algorithms, the algorithms have been tested with these perturbations. These perturbations do not pretend to represent a real world case but they represent some repeatable patterns in which: resources decrease or increase in a fixed way or in which resources temporarily increase or decrease.

In the performed experiments the variated resource is the number of licenses at the second stage of the four stage sequential workflow (see Figure 6.2). With the correlated perturbations each experiment has been subdivided in three intervals, during the middle subdivision the perturbation takes place: the number of licenses available at the second stage is decreased to 10% of its initial (and final) value, or is increase to 10 times the initial value. For example, in an experiment with at maximum 20 licenses, the perturbation reduces the number of licenses from 20 to two (with the decrease perturbation) or it increases the number of licenses from two to 20 (with the increment perturbation).

With the linear perturbations after a number of completed evaluations equals to one third of the total number of evaluations, the perturbation start to take places, it acquire or release one license at time with a fixed period, the period is independent on the other condition of the experiment. Based on the speed at which elaborations proceed certain experiments could end before the perturbation termination while other could terminate after the transient part of the

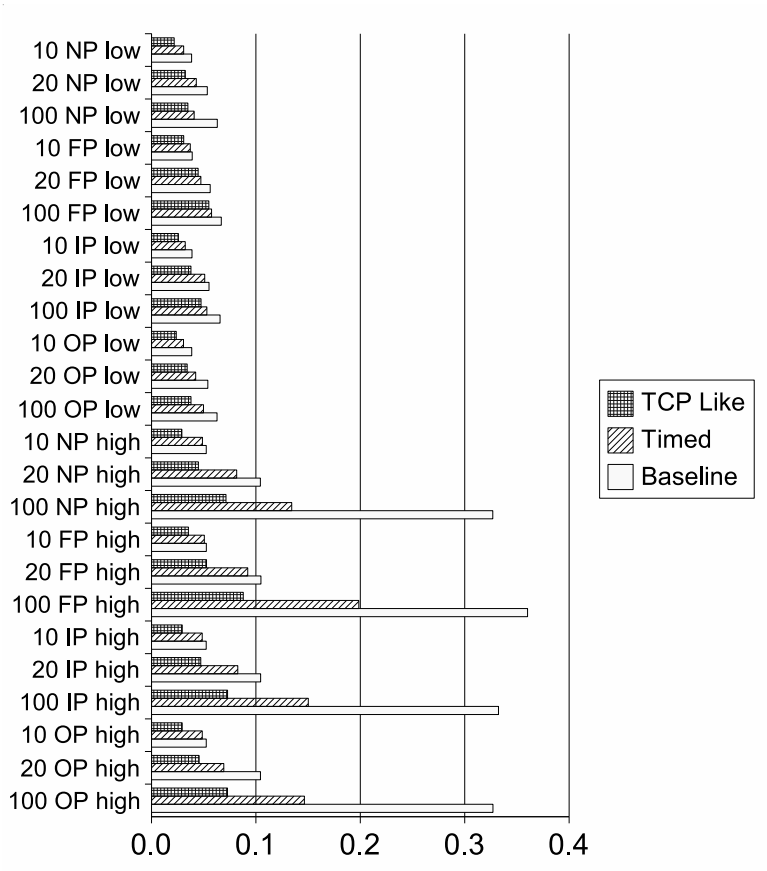

Figure 6.25: Throughput for 2nd stage perturbation: low quality links (up), high quality links (down).

perturbation.

6.5 Controller with Dynamic Environment

This scenario and suite of experiments have been included in [68]. The focus of this section is the ability of controllers to react and adapt to changes in the environment, i.e., to unexpected changes to parameters that may affect performance. The unexpected changes are simulated by a correlated decrease perturbation described in Section 6.4. The setting of the simulation is the same as in previous Section 6.3, each experiment refers to a session composed of 10000 jobs.

In order to interpret the results a baseline for each experiment has been determined, the baseline is defined on an hypothetical controller using a statically defined and immutable multiprogramming level. Each combination of parameters produces a curve throughput vs. MPL and composite service time vs. MPL. These curves enable to identify the throughput-optimal multiprogramming level

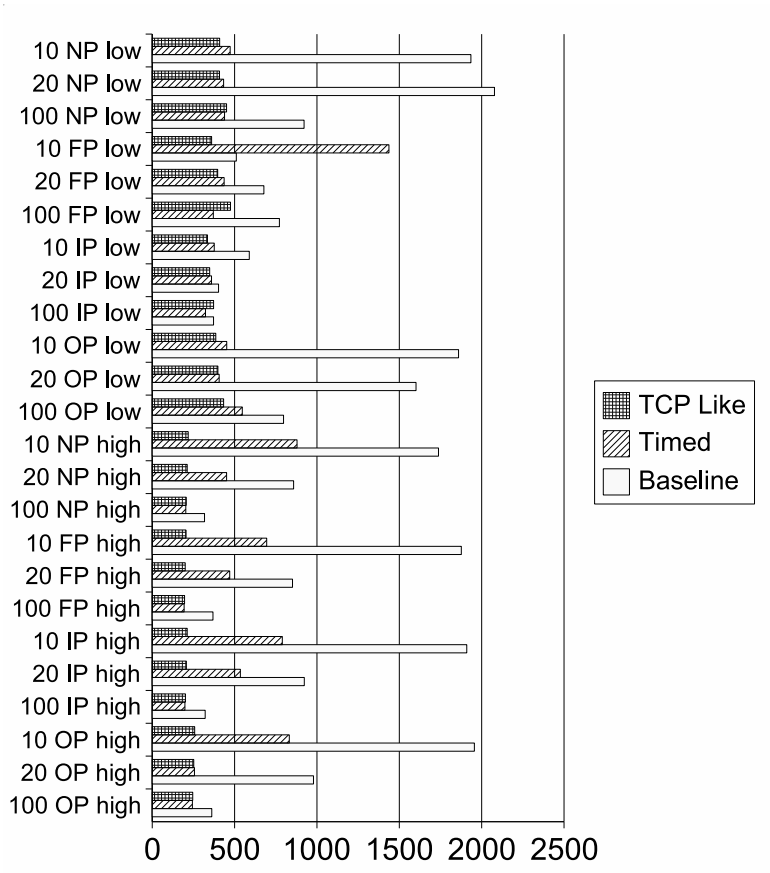

Figure 6.26: Composite service time for 2nd stage perturbation: low quality links (up), high quality links (down).

value for that combination of parameters. In practice, this value cannot be known, as it would require the knowledge of the perturbation that will affect the session. We also remark that the performance baseline is not necessarily the best that can be achieved: it is the best choice only when the multiprogramming level is selected statically.

6.5.1 Results for One Stage Perturbation

Figure 6.25 and 6.26 show throughput and composite service time in all the 24 analyzed scenarios. These scenarios are all the combinations of: link quality (low, high), available licenses (10, 20, 100), and the communication computation overlap patterns (NoPipe, FullPipe, InputPipe, OutputPipe).

It can be seen that, broadly speaking, the controllers are indeed capable of delivering reasonably good throughput even without any apriori knowledge about the system parameters. Moreover, they do so in spite of the substantial perturbation injected into the system.

Table 6.4: Link parameters

Link Speed	Bandwidth Mean [KB/s]	Bandwidth StdDev	Latency Mean [ms]	Latency StdDev
Very High	9770	977	0.01	0.001
High	977	97.7	0.1	0.01
Normal	97.7	9.77	1	0.1

With regard to throughput (Figure 6.25) it can be seen that the TIMED controller obtains always greater throughput than TCP like controller for all scenarios. In many cases the TIMED controller throughput is comparable with the baseline and it reaches the 95% of the baseline for 10 FullPipe for both low and high quality links. For low quality links TCP like controller throughput is always above 60% of the baseline. It can be seen that the benefits of our proposal are less significant for high quality links and when the number of license is high.

Concerning the composite service time, Figure 6.26 shows that for low quality links the TCP like controller composite service time is similar to the TIMED controller composite service time, with the exception of 10 FullPipe they are always lower than the baseline. For high quality links the composite service times obtained by the Timed and the TCP like controller are considerably lower than the baseline, ranging from 10% to 70% of the baseline.

6.6 Complete Suite of Experiments

This scenario and suite of experiments have been included in [71]. To improve and to further test controller algorithms behavior, the suite of experiments have been augmented to include all the four perturbations described in Section 6.4 and all the four controller algorithms described in Section 5.3.

The test case is similar to the one described in the previous sections. We considered a workflow schema composed of four identical services serially connected (Figure 6.2). Each experiment is characterized by the following parameters: *i)* communication computation overlap pattern: NoPipe, InputPipe, FullPipe or OutputPipe; *ii)* number of licenses, either 10 or 100; *iii)* links quality: Normal, High or Very High (see Table 6.4); *iv)* perturbation: linear increase, linear decrease, correlated increase or correlated decrease. The remaining parameters are summarized in Table 6.5, buffer cache size is expressed in number of blocks. Each experiment refers to a session composed of 1000 jobs. It is assumed that at any time, there is always at least one job submitted at the composite service and waiting to be executed (except when the session is about to complete).

Table 6.5: Working point

Parameter	Value
Buffer Cache Size	100000
Minimum Disk Read Time [s]	0.001
Maximum Disk Read Time [s]	0.002
Minimum Disk Write Time [s]	0.0002
Maximum Disk Write Time [s]	0.0004
Sector Size [KB]	64
Initial Data Size [MB]	64
Start Up Time [s]	30
Elaboration Speed [B/s]	65536
Output/Input Ratio	1.1

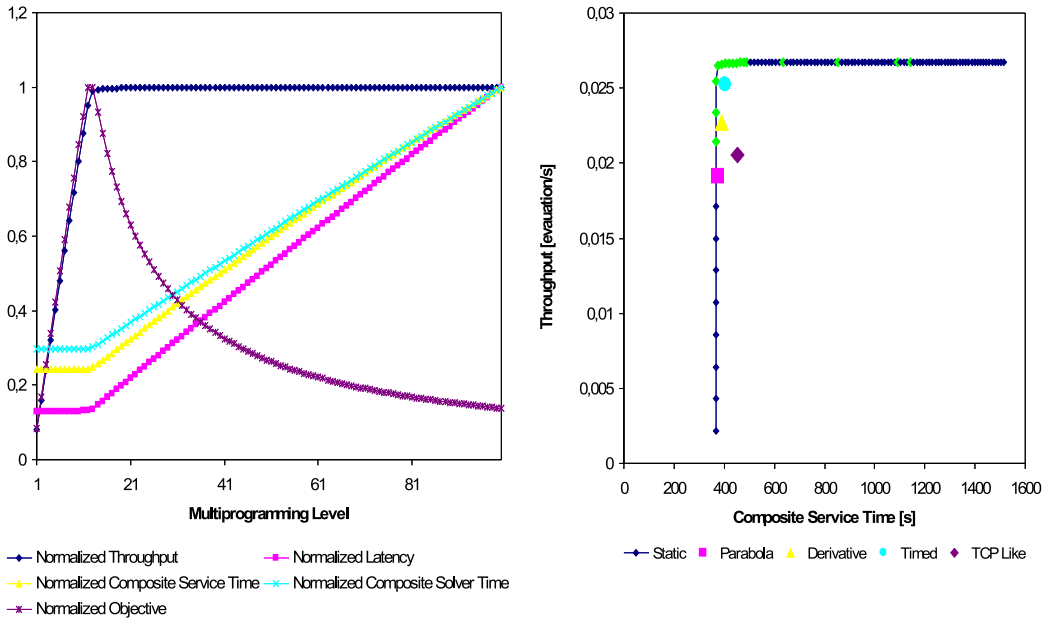

Figure 6.27: Performances in static environment (10 licenses, Very High quality links, FullPipe.)

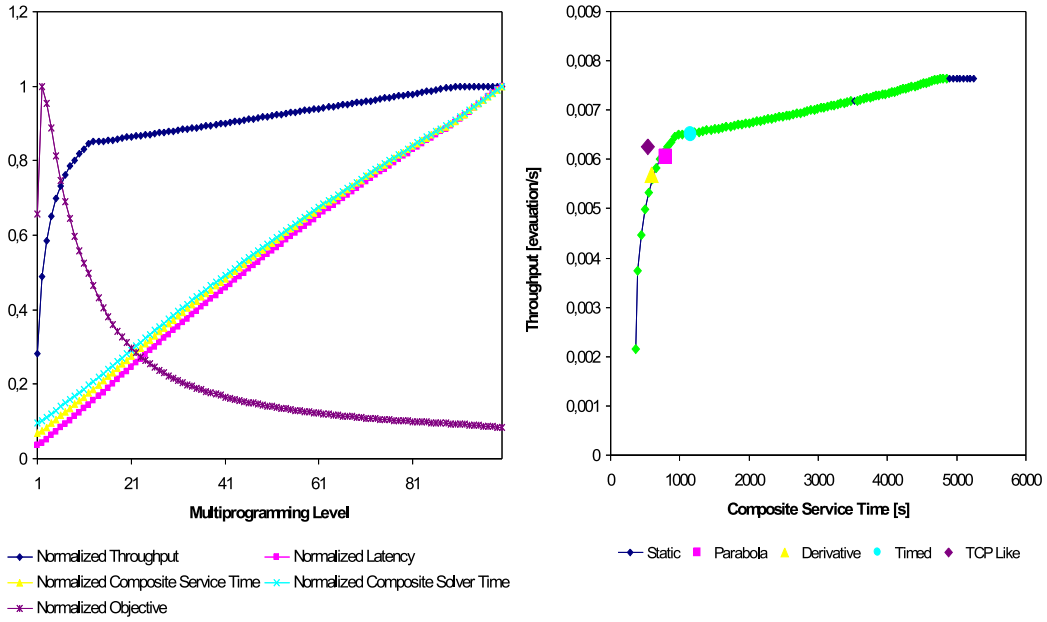

Figure 6.28: Performances in environment with correlated decrease perturbation (10 licenses, Very High quality links, FullPipe.)

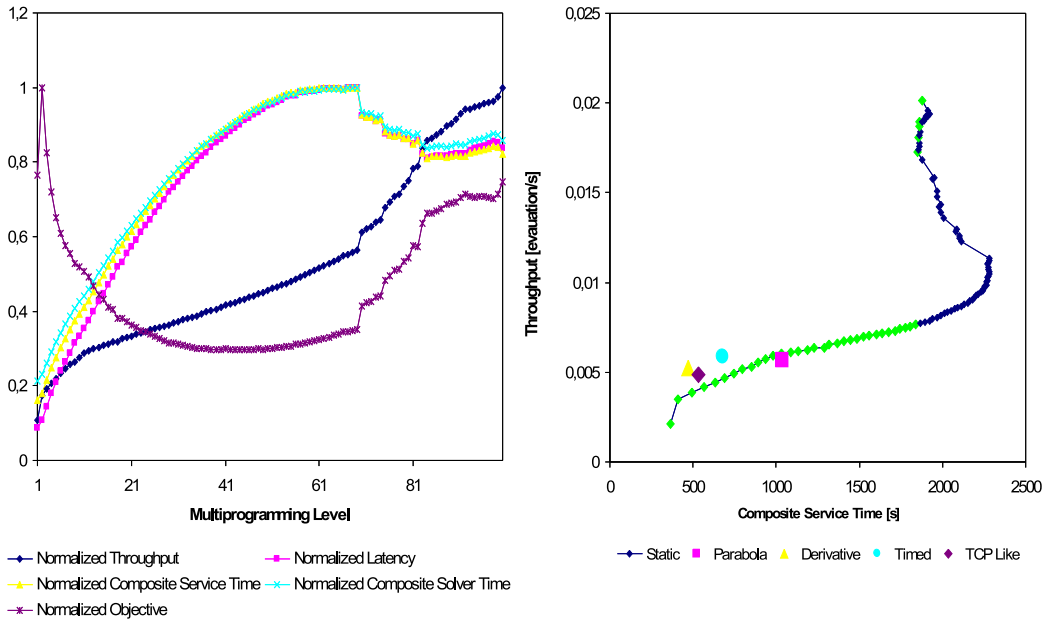

Figure 6.29: Performances in environment with linear decrease perturbation (10 licenses, Very High quality links, FullPipe.)

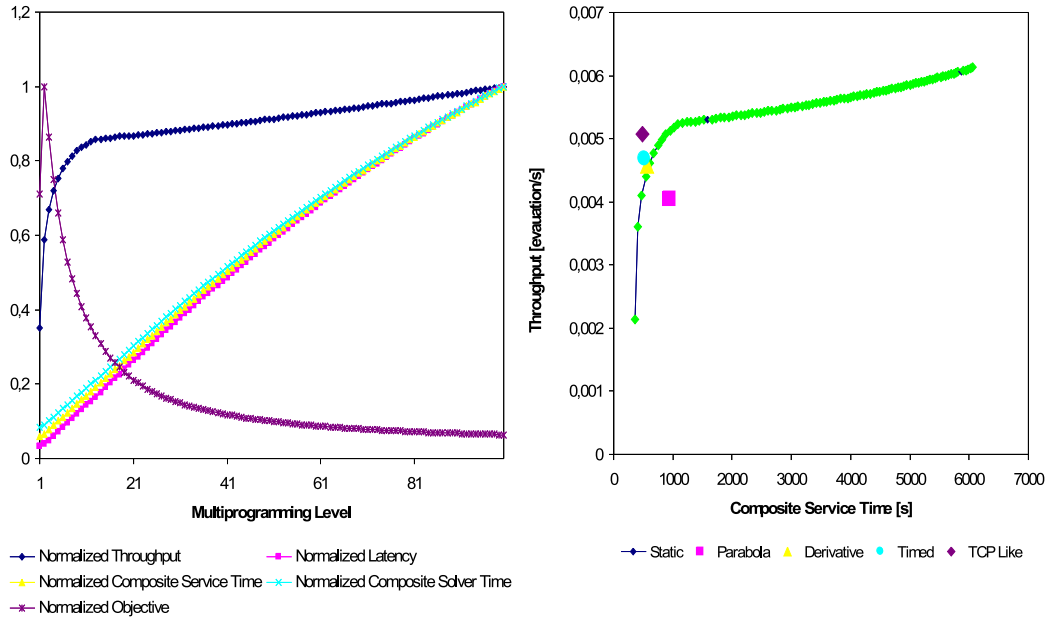

Figure 6.30: Performances in environment with correlated increase perturbation (10 licenses, Very High quality links, FullPipe.)

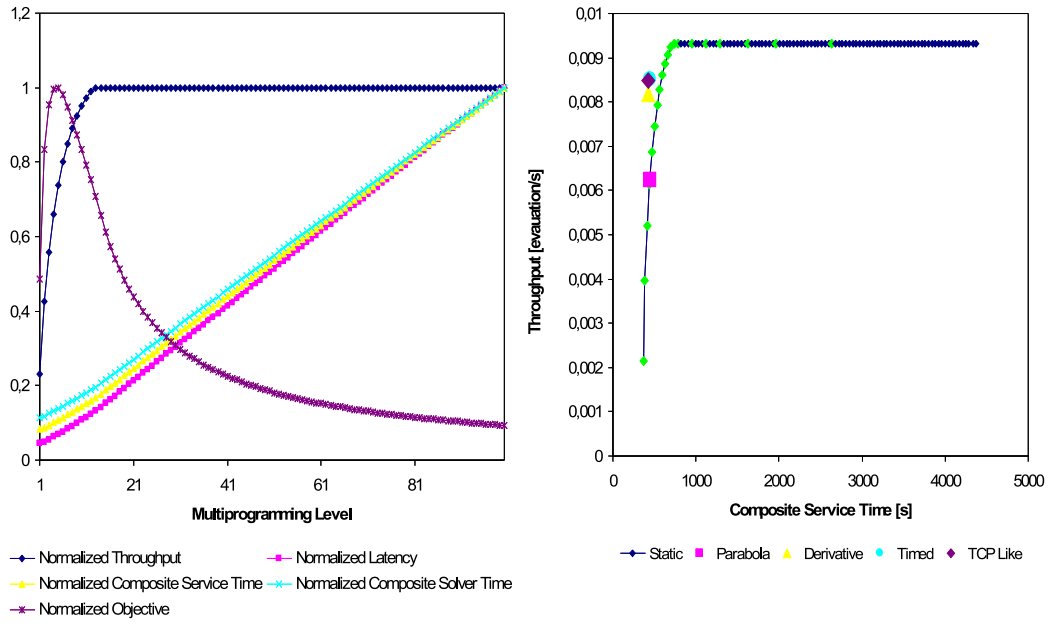

Figure 6.31: Performances in environment with linear increase perturbation (10 licenses, Very High quality links, FullPipe.)

For the sake of brevity we show only the results for the combination of 10 licenses, with links of Very High quality and FullPipe communication computation overlap pattern. Results for the other combinations of parameters are similar to those described in the previous sections.

Figures 6.27, 6.28, 6.29, 6.30, and 6.31 contains two charts. The chart on the left plots the considered performance indexes (throughput, latency, composite service time, composite solver time, and objective index) against a static multiprogramming level. The chart on the right is a scatter chart plotting the throughput against the composite service time, green dots represent multiprogramming level values belonging to the pareto frontier. Furthermore, the scatter chart contains a point for each controller algorithm, this point represents the performance of the algorithm in terms of final throughput and service composite time.

Both for the static case (Figure 6.27) and the cases affected by perturbation (Figures 6.28, 6.29, 6.30, and 6.31), it can be seen that because of the very high links quality, the throughput does not denote any trashing, i.e. throughput curves are almost always monotonic. Other performance indexes behave as described in Section 5.1.

Results for the correlated decrease (see Figure 6.23) and correlated increase (see Figure 6.24) are quite similar both in terms of scale and shape. With the correlated decrease perturbation, throughput ranges from *2,1E-3* to *7,7E-3* and service composite time ranges from *3,7E2* to *5,2E3*, while with the correlated increase perturbation, throughput ranges from *2,1E-3* to *6,1E-3* and composite service time ranges from *3,7E2* to *6,1E3*. Because for these two perturbations the amount of resources is equivalent for *2/3* of the simulated sessions, this similarity is not totally unexpected.

The most interesting case, shown in Figure 6.29, is with the linear decrease perturbation. With this perturbation, the number of licenses, for executing the solver at the second stage of the workflow, are decreased at a constant rate. At the beginning of the session there are much more resources than at the end. Quite surprisingly, composite service time is no more monotonic, this means that a higher constant multiprogramming level could lead to a greater throughput and a lesser composite service time. This perturbation makes more difficult the manual choice of a good multiprogramming level. As stated in Chapter 5, medium values of multiprogramming level usually give a good trade-off between the reduction of the cost and the speed up, but in this case they give the worst results.

As shown in Figure 6.27, because of their transient and dynamic behavior, controllers cannot outperform a constant multiprogramming level in a static environment. Constant multiprogramming level is always the best choice in a static environment. On the other hand inter-organizational and Internet-based appli-

cations do not deal with static environments.

Table 6.6: Dominating multiprogramming levels, for 10 licenses, Very High quality links and FullPipe overlap pattern.

Perturbation	Parabola	Derivative	TIMED	TCP Like
Static	5	4	4	9
Correlated Decrease	1	0	3	0
Linear Decrease	3	0	0	0
Correlated Increase	7	0	0	0
Linear Increase	0	0	0	0

Table 6.7: Dominated multiprogramming levels, for 10 licenses, very high quality links and FullPipe overlap pattern.

Perturbation	Parabola	Derivative	TIMED	TCP Like
Static	0	0	0	0
Correlated Decrease	0	1	0	6
Linear Decrease	0	6	6	3
Correlated Increase	0	0	2	6
Linear Increase	0	4	4	5

With respect to each controller algorithms and for each scenario, we define the *number of dominated MPLs* as the number of multiprogramming levels that exhibits a throughput lower and a composite service time greater than the controller. Similarly we define the *number of dominating MPLs* as the number of multiprogramming levels that exhibits a throughput greater and a composite service time lower than the controller.

Table 6.6 shows the number of dominating MPLs for each combination of controller algorithm and perturbation. Especially in the static environment, as stated above, we expect a certain number of MPLs be able to dominate the controllers algorithms. For example, there are five levels of multiprogramming dominating the Parabola controller, four dominating both the TIMED and the Derivative controllers, and nine dominating the TCP Like controllers. When pertubations affect the system performances, the Parabola controller is still dominated by constant MPLs for all perturbations but linear decrease, this algorithm suffers a sort of overfitting. Other controllers start to behave as expected and, excluding the TIMED controller with the correlated decrease perturbation, no constant multiprogramming level is able to dominate the algorithms.

Table 6.7 shows the number of dominated MPLs for each combination of controller algorithm and perturbation. Again, in the static environment, controllers

are not able to dominate constant multiprogramming levels. Moving toward experiments with perturbations, the controller algorithms start to dominate some multiprogramming levels.

Conclusion

Organizational-wide and inter-organizational computing systems cannot rely on human tuning and should be able to tune themselves to maximize performance on a continuous basis and in a non-intrusive fashion, monitoring the system load and performance. To do so underlying middleware should be based on autonomic management. Autonomic management encompasses several attributes such as self-healing, self-provisioning, self-optimizing (load-balancing), and self-configuring. Ideally, human administrators only need to set high level goals in form of utility functions to hint the system in which dimensions to maximize first.

The scheduling of jobs in application-level workflows may have substantial effects on performance. Workflows resulting from the Internet-based aggregation of resources exported by multiple organizations are especially challenging from this point of view. The task is complicated further by the very same nature of this scenario, in which there are many performance critical parameters whose values can hardly be known in advance and that can change dynamically and unpredictably.

Additional workloads injected by other competing activities are just one of the many possible causes for this dynamism. In this thesis the above scenario has been addressed by proposing job scheduling policies capable of controlling job scheduling adaptively, based on simple measures on the current status of the system. These policies are based on a mechanism simple to deploy in practice that, in particular, does not need any hook from the organizations participating in the workflow.

This proposal has been evaluated in detail, by simulation, and this work proves that it is indeed capable of finding automatically a suitable trade-off between throughput and cost, in spite of the fact that both parameters and their variations are unknown. Although this analysis needs to be broaden toward further forms of perturbation, such results are quite encouraging. This proposal may be an effective way to cope efficiently and automatically with the myriad of performance-critical factors that may be unknown, may vary unpredictably and are beyond the control of the scheduling machinery.

Bibliography

[1] W. Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. [cited at p. 27]

[2] S. Abdelwahed, N. Kandasamy, and S. Neema. A control-based framework for self-managing distributed computing systems. In *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 3–7, New York, NY, USA, 2004. ACM Press. [cited at p. 18, 19]

[3] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):80–96, 2002. [cited at p. 17, 18]

[4] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Verlag, 2004. [cited at p. 9]

[5] G. Alonso, B. Reinwald, and C. Mohan. Distributed data management in workflow environments. In *7th International Workshop on Research Issues in Data Engineering (RIDE '97) High Performance Database Management for Large-Scale Applications*, page 82, Los Alamitos, CA, USA, 1997. IEEE Computer Society. [cited at p. 35]

[6] J. April, F. Glover, J. P. Kelly, and M. Laguna. Simulation-based optimization: practical introduction to simulation optimization. In *WSC '03: Proceedings of the 35th Winter simulation conference*, pages 71–78, 2003. [cited at p. 8]

[7] W. Y. Arms, S. Aya, M. Calimlim, J. Cordes, J. Deneva, P. Dmitriev, . G. Johannes Gehrke2, L. Gibbons, C. D. Jones, V. Kuznetsov, D. Lifka, M. Riedewald, D. Riley, A. Ryd, and G. J. Sharp. Three case studies of large-scale data flows. In *ICDEW'06: 22nd International Conference on Data Engineering Workshops*, pages 66–75, Los Alamitos, CA, USA, 2006. IEEE Computer Society. [cited at p. 7]

[8] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete–Event System Simulation*. Prentice–Hall, Upper Sadder River, New Jersey, 4d edition, 2005. [cited at p. 8]

[9] A. Bartoli, R. Jiménez-Peris, B. Kemme, C. Pautasso, S. Patarin, S. Wheater, and S. Woodman. The adapt framework for adaptable and composable web services. *IEEE Distributed Systems On Line*, Sep 2005. Web Systems Section. [cited at p. 7]

[10] BEA, IBM, Microsoft, SAP AG and Siebel Systems. *Business Process Execution Language for Web Services Version 1.1*, 2003. http://www.ibm.com/developerworks/library/specification/ws-bpel/. [cited at p. 28]

[11] V. Bhat, M. Parashar, H. Liu, N. Kandasamy, M. Khandekar, S. Klasky, and S. Abdelwahed. A self-managing wide-area data streaming service. *Cluster Computing*, 10(4):365–383, Dec 2007. [cited at p. 18, 19, 20, 35, 36]

[12] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005. [cited at p. 29]

[13] K. Botros, D. Sennhauser, K. Jungowski, G. Poissant, H. Golshan, and J. Stoffregen. Effects of dynamic penalty parameters on the convergence of moga in optimization of a large gas pipeline network. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Aug. 2004. [cited at p. 8]

[14] E. K. Boulougouris, A. D. Papanikolaou, and G. Zaraphonitis. Optimization of arrangements of ro-ro passenger ships with genetic algorithms. *Ship Technology Research*, 51(3):99–105, 2004. [cited at p. 8]

[15] M. J. Carey, S. Krishnamurthi, and M. Livny. Load control for locking: the "half-and-half" approach. In *PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 72–84, New York, NY, USA, 1990. ACM Press. [cited at p. 15]

[16] Y. Carson and A. Maria. Simulation optimization: methods and applications. In *WSC '97: Proceedings of the 29th Winter simulation conference*, pages 118–126, New York, NY, USA, 1997. ACM Press. [cited at p. 23]

[17] H. Casanova, A. Legrand, and M. Quinson. Simgrid: a generic framework for large-scale distributed experimentations. In *UKSIM/EUROSIM '08: Proceedings of the 10th IEEE International Conference on Computer Modelling and Simulation*, 2008. [cited at p. 49, 50, 53]

[18] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 349, Washington, DC, USA, 2000. IEEE Computer Society. [cited at p. 13]

[19] F. Casati, S. Ilnicki, L. jie Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eflow. In *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 13–31. Springer-Verlag, 2000. [cited at p. 7]

[20] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda. Decentralized orchestration of composite web services. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 134–143, New York, NY, USA, 2004. ACM Press. [cited at p. 14]

[21] H. Chen and P. Mohapatra. Session-based overload control in QoS-aware Web servers. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2002)*, volume 2, pages 516–524. IEEE, June 2002. [cited at p. 17]

[22] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. *The WS-Resource Framework*, 2004. http://www-106.ibm.com/developerworks/library/wsresource/ws-wsrf.pdf. [cited at p. 20]

[23] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003. [cited at p. 29]

[24] DESMO–J. *The DESMO–J homepage*, 2000. http://www.desmoj.de. [cited at p. 49]

[25] D. Dyachuk and R. Deters. Optimizing performance of web service providers. In *21st International Conference on Advanced Networking and Applications (AINA '07)*, volume 0, pages 46–53, Los Alamitos, CA, USA, 2007. IEEE Computer Society. [cited at p. 16]

[26] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 276–286, New York, NY, USA, 2004. ACM Press. [cited at p. 17]

[27] Esteco. *modeFRONTIER*, 2007. [cited at p. 8]

[28] Y. Fu, B. Kachnowski, and E. Lee. Occupant model correlation using a multiobjective evolution strategy. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2004. [cited at p. 8]

[29] A. Gaiddon, D. D. Knight, and C. Poloni. Multicriteria design optimization of a supersonic inlet based upon global missile performance. *Journal of Propulsion and Power*, 20(3):542–558, 2004. [cited at p. 8]

[30] A. Giassi, F. Bennis, and J. J. Maisonneuve. Multidisciplinary design optimisation and robust design approaches applied to concurrent design. *Structural and Multidisciplinary Optimization*, 28(5):356–371, 2004. [cited at p. 8]

[31] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, Dec. 2007. [cited at p. 7, 9, 20]

[32] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to web services architecture. *IBM Sys. Journal*, 41:170–177, 2002. [cited at p. 9, 20]

[33] GridSim. Gridsim: A grid simulation toolkit for resource modeling and application scheduling for parallel and distributed computing. http://www.gridbus.org/gridsim/, 2008. [cited at p. 53]

[34] H.-U. Heiss and R. Wagner. Adaptive load control in transaction processing systems. In *VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases*, pages 47–54, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. [cited at p. 15, 16, 17, 47]

[35] E. Hepp, O. Lohne, and S. Sannes. Extended casting simulation for improved magnesium die casting. *Magnesium: Proceedings of the 6th International Conference on Magnesium Alloys and Their Applications*, pages 669–674, Apr. 2005. [cited at p. 8]

[36] R. Jiménez-Peris, M. Patiño-Martínez, and B. Kemme. Enterprise grids: Challenges ahead. *Journal of Grid Computing*, 5(3):283–294, Sep. 2007. [cited at p. 36]

[37] W. E. Johnston. Semantic services for grid-based, large-scale science. *Intelligent Systems*, 19(1):34–39, 2004. [cited at p. 29]

[38] M. Karlsson and M. Covell. Dynamic black-box performance model estimation for self-tuning regulators. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 172–182, Washington, DC, USA, 2005. IEEE Computer Society. [cited at p. 18]

[39] Y. Kitatsuji, K. Yamazaki, H. Koide, M. Tsuru, and Y. Oie. Influence of network characteristics on application performance in a grid environment. *Telecommunication Systems*, 30(1-3):99–121, Nov 2005. [cited at p. 14]

[40] G. Kola, T. Kosar, J. Frey, M. Livny, R. Brunner, and M. Remijan. Disc: A system for distributed data intensive scientific computing. In *Proc. of First Workshop on Real, Large Distributed Systems, December 2004.*, Dec 2004. [cited at p. 7]

[41] T. Kosar and M. Livny. Stork: making data placement a first class citizen in the grid. In *ICDCS'04: Proceedings of the 24th Internation Conference on Distributed Computing Systems*, pages 342–349, 2004. [cited at p. 35]

[42] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance - Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., 1984. [cited at p. 50]

[43] F. Leymann. The (service) bus: Services penetrate everyday life. In B. Benatallah, F. Casati, and P. Traverso, editors, *ICSOC 2005: Third International Conference on Service-Oriented Computing*, 2005. [cited at p. 19]

[44] D. Liu, J. Peng, K. H. Law, G. Wiederhold, and R. D. Sriram. Composition of engineering web services with distributed data-flows and computations. *Advanced Engineering Informatics*, 19:25–42, 2005. [cited at p. 35]

[45] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *RTAS '01: Proceedings of the Seventh Real-Time Technology and Applications Symposium*, page 51, Washington, DC, USA, 2001. IEEE Computer Society. [cited at p. 17, 18]

[46] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006. [cited at p. 29]

[47] B. Ludäscher, I. Altintas, and A. Gupta. Compiling abstract scientific workflows into web service workflows. In *Conference on Scientific and Statistical Database Management, 2003. 15th International*, pages 251–254, 9-11 July 2003. [cited at p. 29]

[48] J. J. Maisonneuve, S. Harries, J. Marzi, H. C. Raven, U. Viviani, and H. Piippo. Towards optimal design of ship hull shapes. In *Proceedings of the 8th International Marine Design Conference*, pages 31–42, 2003. [cited at p. 8]

[49] J. M. Milan-Franco, R. Jiménez-Peris, M. P. no Martínez, and B. Kemme. Adaptive middleware for data replication. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 175–194, New York, NY, USA, 2004. Springer-Verlag New York, Inc. [cited at p. 16]

[50] Oasis. *UDDI Committee Specification*, 2002. `http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm`. [cited at p. 20]

[51] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing. *Commun. ACM*, 46(10):24–28, 2003. [cited at p. 27]

[52] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, Nov. 2007. [cited at p. 9]

[53] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Syst.*, 23(1-2):127–141, 2002. [cited at p. 18]

[54] K. Pingali and P. Stodghill. A distributed system based on web services for computational science simulations. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 297–306, New York, NY, USA, 2006. ACM Press. [cited at p. 7, 14]

[55] S. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. In *WWW2002: Proceedings of the 11th International WWW Conference*, 2002. [cited at p. 7]

[56] D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*. John Wiley and Sons, West Sussex, England, 1997. [cited at p. 22, 24]

[57] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Design and evaluation of load control in web server systems. In *Proceedings of American Control Conference*, 2004. [cited at p. 17, 18]

[58] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, and A. Wierman. How to determine a good multi-programming level for external scheduling. In *ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering*, Los Alamitos, CA, USA, April 2006. IEEE Computer Society. [cited at p. 16]

[59] G. Singh, C. Kesselman, and E. Deelman. Optimizing grid-based workflow execution. *Journal of Grid Computing*, 3(3):201–219, 2005. [cited at p. 7, 14, 40]

[60] B. Srivastava and J. Koehler. Web service composition current solutions and open problems. In *In Proceedings of ICAP 03*, 2003. [cited at p. 7]

[61] M. Stal. Web services: beyond component-based computing. *Communications of the ACM*, 45(10):71–76, 2000. [cited at p. 9]

[62] J. R. Swisher, P. D. Hyden, S. H. Jacobson, and L. W. Schruben. A survey of simulation optimization techniques and procedures. In J. Joines, R. Barton, K. Kang, and P. Fishwick, editors, *WSC '00: Proceedings of the 32th Winter Simulation Conference*, pages 119–128, December 2000. [cited at p. 8]

[63] I. Taga, A. Funakubo, and Y. Fukui. Design and development of an artificial implantable lung using multiobjective genetic algorithm: Evaluation of gas exchange performance. *ASAIO Journal*, 51(1):92–102, 2005. [cited at p. 8]

[64] G. Teodoro, T. Tavares, R. Ferreira, T. Kurc, W. M. Jr., D. Guedes, T. Pan, and J. Saltz. A run-time system for efficient execution of scientific workflows on distributed environments. In *SBAC-PAD '06: Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'06)*, pages 81–90, Washington, DC, USA, 2006. IEEE Computer Society. [cited at p. 7, 14]

[65] Transaction Processing Performance Council. *TPC-App*, 2007. http://www.tpc.org/tpc_app/default.asp. [cited at p. 16]

[66] F. Vavak and T. C. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *ICEC '96: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 192–195. IEEE Press, 20-22 May 1996. [cited at p. 62]

[67] P. Vercesi and A. Bartoli. On the performance of inter-oganizational design optimization systems. In *WSC '06: Proceedings of the 38th Winter simulation conference*, pages 1177–1186, 2006. [cited at p. 24, 35, 53]

[68] P. Vercesi and A. Bartoli. Adaptive performance control of internet-based grids in a dynamic environment. In *ICCCN 2007: Proc. of 16th International Conference on Computer Communications and Networks, Workshop on Advanced Networking*, 2007. [cited at p. 70]

[69] P. Vercesi and A. Bartoli. Adaptive performance tuning for internet-based workflows. In *COMPSAC 2007: Proc. of 31st Annual IEEE International Computer Software and Applications Conference (COMPSAC 2007)*, 2007. [cited at p. 62]

[70] P. Vercesi and A. Bartoli. Performance-related issued in internet-based integration of cae systems. In *MITIP 2007: Proc. of the 9th International Conference on the Modern Information Technology in the Innovation Process of the Industrial Enterprises*, Sep. 2007. [cited at p. 36]

[71] P. Vercesi and A. Bartoli. Runtime performance optimization of grid-workflows execution. *In preparation for submission to international journal*, 2008. [cited at p. 72]

[72] W3C. *Web Services Description Language (WSDL) 1.1*, 2001. http://www.w3.org/TR/wsdl. [cited at p. 19, 20]

[73] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *Proceedings USENIX of the 2003.*, 2003. [cited at p. 17]

[74] J. Wildstrom, P. Stone, E. Witchel, and M. Dahlin. Machine learning for on-line hardware reconfiguration. In *IJCAI-07: Proceedings of the 20th international joint conference on artifical intellingence*, pages 1113–1118, 2007. [cited at p. 18, 19]

[75] Workflow Management Coalition. *Workflow Management Coalition Terminology & Glossary.* Workflow Management Coalition, 1999. Document No. WFMC-TC-1011.3. [cited at p. 27]

[76] P. Young. *Recursive estimation and time-series analysis: an introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1984. [cited at p. 47]

List of Figures

List of Tables