AUTOMATIC ENABLEMENT, COORDINATION AND RESOURCE USAGE PREDICTION OF UNMODIFIED APPLICATIONS ON CLOUDS

By

ANDRÉA MATSUNAGA

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2010

To everyone who contributed to my education, who shared experiences with me, who helped making this dissertation a reality

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. José Fortes, for his generous guidance and support throughout my many memorable years as graduate student. Dr. Fortes gifted me with freedom and opportunities to pursue exciting and challenging problems, guided me through the art of scientific writing, offered access to the entire computational infrastructure at the ACIS laboratory, and always made himself available to share his expertise despite his cruel schedule. It has truly been an honor.

I am also thankful to Dr. Renato Figueiredo, Dr. Herman Lam and Dr. Joseph Wilson, members of my supervisory committee, for all the comments and suggestions that helped improve this dissertation.

Many have influenced directly this dissertation. In particular, I would like to thank Dr. Kate Keahey and Tim Freeman for their support with Nimbus clouds and in arranging resources for my many long jobs; Dr. Leonid Moroz for introducing me to BLAST and providing input sequences, Dr. William Farmerie for bringing to my attention the challenges in dealing with large datasets produced by 454 and newer sequencing technologies; Dr. Haijun Zhu for presenting RAxML to me; Mr. Michael Ott for providing RAxML inputs; Dr. Violetta Cavalli-Sforza, Dr. Wayne Ward and Dr. Stanley Su for their patience in explaining the installation, configuration and usage of their applications; and anonymous reviewers for their feedback.

Others brought balance during this long journey. I would like to thank my family, friends, past and present ACIS laboratory colleagues and staff, and members of the Transnational Digital Government team; especially Dr. Sumalatha Adabala, Dr. Maurício Tsugawa and Dr. Márcia Rupnow, for all the encouragement and generous help.

Last but not least, this work would not have been possible without the financial support from: the National Science Foundation Grants EIA-9975275, EIA-0224442, ACI-0219925, EEC-

0228390, EIA-0107686, EIA-0131886, OCI-0721867, CNS-0821622, CMMI-0742580, and IIP-0758596; the NSF Middleware Initiative (NMI) collaborative grants ANI-0301108/ANI-0222828, and SCI-0438246; the Interdisciplinary Center for Biotechnology Research; the industrial members of the NSF Center for Autonomic Computing (Intel, Northrop-Grumman, and Microsoft); the BellSouth Foundation; the IBM SUR grants; the VMware Corporation and the equipment from Cyberguard.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

AUTOMATIC ENABLEMENT, COORDINATION AND RESOURCE USAGE PREDICTION OF UNMODIFIED APPLICATIONS ON CLOUDS

By

Andréa Matsunaga

May 2010

Chair: José A. B. Fortes
Major: Electrical and Computer Engineering

As computing paradigms evolve and application demands grow, users face challenges in efficiently accessing, using and coordinating the large number of heterogeneous resources of complex computing systems. In the context of cloud computing, this work proposes methods to automate time-consuming processes currently performed by end users or developers, namely, the enabling of applications as services, the scaling-out of applications, and the estimation of resource consumption by applications.

First, offering application services to consumers requires significant efforts by cloud providers, since many existing applications, useful to consumers, are not implemented as Web Services. To speed the creation of application services, this dissertation describes an approach to the automatic enablement of existing text-based applications with a command-line interface, called Command-Line Application Wrapper Service (CLAWS). Compared to other application-wrapping approaches, CLAWS exposes a simpler interface to users, completely hiding the complexities of understanding, developing and deploying access-controlled Web Services. CLAWS is motivated by, and effective for, the important case of interactive applications, which has not been considered by other approaches. The use of CLAWS is evaluated in the context of a

transnational digital government project, where CLAWS greatly facilitated the integration of translation and conversation applications into an information sharing system.

Second, in order to address the need to scale computationally intensive applications on clouds, this dissertation presents an end-to-end approach and best practices to run such large applications on multiple clouds. The techniques were implemented in CloudBLAST, combining the use of MapReduce model for coordinating the parallel execution of unmodified applications, machine virtualization for encapsulating applications and their execution environments, network virtualization to connect resources behind firewalls/NATs, and cloud management services to coordinate the deployment of a virtual cluster on demand. Experiments with CloudBLAST in cloud testbeds have demonstrated good scalability executing a bioinformatics application (BLAST), even when using cloud resources across wide-area networks and in the presence of machine and network virtualization.

Third, a useful piece of information for scheduling jobs, typically not available, is the extent to which applications will use available resources, once executed on heterogeneous clouds. This dissertation comparatively assesses the suitability of several machine learning techniques for predicting linear and non-linear spatiotemporal utilization of resources, taking into account application- and system-specific attributes. This work also extends an existing classification tree algorithm, called Predicting Query Runtime (PQR), to the regression problem by selecting the best regression method for each collection of data on the leaves. When compared to other regression algorithms (linear, k-nearest neighbor, decision table, radial basis function network, and support vector machine), the new method (PQR2) yields the best mean percentage error predicting execution time, memory and disk consumption for two bioinformatics applications, BLAST and RAxML, deployed on scenarios that differ in system and usage.

CHAPTER 1
INTRODUCTION

Cloud Computing

The explosion of software applications and data combined with high-speed ubiquitous Internet and large datacenters motivated the development of *cloud computing* to make it easy for users to take advantage of the vast computational power available in today's servers and data centers. While imprecisely defined [14][68][135], cloud computing broadly refers to the use of managed distributed resources to deliver services to multiple users on demand, often using virtualization to provide execution environments as needed by applications and users [34]. The vision, shared with Grid and utility computing, is for users to be able to access and pay for computational services just as conveniently as when using electrical appliances powered by the electrical grid.

Three main types of service can be delivered by cloud providers according to the level of abstraction (Figure 1-1):

- *Infrastructure-as-a-Service (IaaS)*: end users get access to raw CPU cycles, storage and network, often in the form of virtual machines (VM) instantiated from appliances (VM images) supplied by the end user. IaaS providers offer a high degree of freedom for end users in the selection and configuration of the necessary execution environment. Conversely, the complexity of designing the system well also falls under the end user's responsibility. This flexibility has attracted customers from various fields to commercial IaaS cloud providers, such as 3tera [1], Amazon Elastic Compute Cloud (EC2) [7], Amazon Simple Storage Service (S3) [8], Engine yard [41], GoGrid [114], and Joyent [70]. On the academic front, various institutions started to offer IaaS to the scientific community on a voluntary basis in 2008, forming the Science Clouds [54].

- *Platform-as-a-Service (PaaS)*: offers a specific development and execution environment to end users, possibly on top of IaaS clouds. A typical environment is composed of an operating system, programming applications and data management tools. When selecting a PaaS cloud provider, end users need to evaluate potential vendor lock-in[1] and investigate the level of freedom in developing, deploying and testing the application. Notable

[1] The vendor lock-in concern is also present in IaaS and SaaS. However, while PaaS providers may include proprietary libraries in the platform, it is common for multiple IaaS providers to support a certain VM format, and for multiple SaaS providers to support standard data formats (e.g., e-mail, documents, and photos).

commercial PaaS cloud providers include Google App Engine [57], Microsoft Windows Azure Platform [89], Cycle Computing CycleCloud [30], and Salesforce Force.com Platform [110].

- *Software-as-a-Service (SaaS)*: delivers very specific applications to a broad user base, often in the form of a web application. The applications can include business processes, scientific experiments, collaborative tools, and even entertainment services. End users can often configure the application as desired and supply their own data, but modifications or expansions of the application are not permitted, since the application has been architected by the provider to scale on demand to a large audience. Prominent SaaS cloud providers include Google apps [58], Microsoft Online Services [87], and Salesforce.com [111].

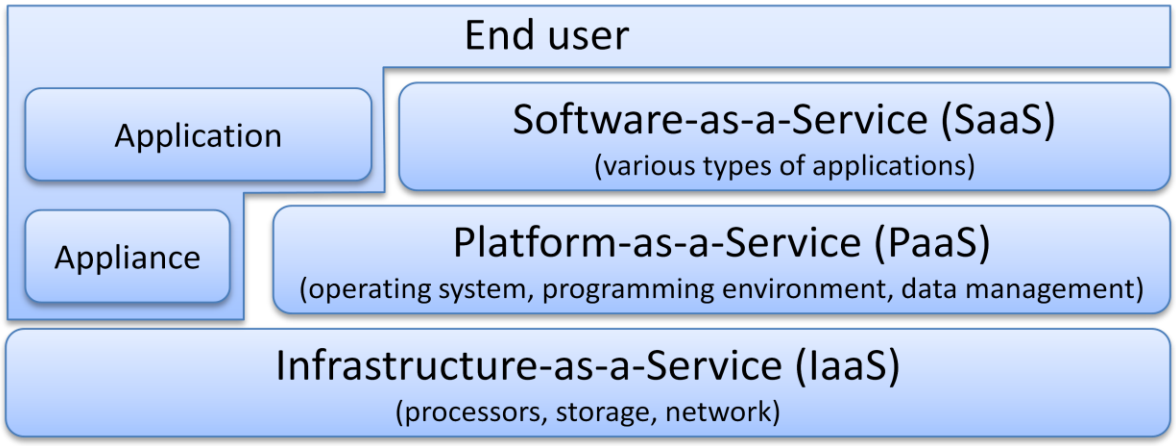


Figure 1-1. Types of services delivered by cloud providers to the end user according to the level of abstraction. IaaS providers offer access to raw resources (processors, storage and network), PaaS providers offer application development, deployment and test environments, whereas SaaS providers offer ready-to-use applications. End users can purchase and combine different services with varying degrees of freedom.

With respect to accessibility, a cloud can be classified as:

- *Public cloud:* the general public can access computational services over the Internet. Since many users may share the same infrastructure, cloud providers are responsible for providing the necessary isolation between different users and for clearly stating the service qualities through Service Level Agreements (SLAs).

- *Private (or internal) cloud:* only a group or an organization can access on-premise computational services. When the level of privacy offered by public clouds is unacceptable or poorly implemented, some organizations opt for creating a private cloud, increasing the level of security by physically isolating resources and allowing own security policies to be enforced. Although many of the benefits of cloud computing (e.g., scalability, reliability, and agility) can be achieved in a private cloud, expenditures of acquiring, managing and operating an Information Technology (IT) infrastructure cannot be avoided, making it hard

for organizations with small and medium capital and workload to exploit economies of scale.

- *Hybrid cloud:* provides a computational service by combining public and private clouds. The private cloud handles the sensitive parts of the application, while the remaining parts of the application can be delegated to public clouds. In this work, the term *sky computing* denotes the use of resources across multiple public and/or private clouds connected through wide-area networks to provide on demand services.

Key qualities that have stimulated the recent interest in cloud computing include:

- *Elasticity:* the ability to increase or decrease the amount of resources allocated to a task on demand.

- *Reliability:* the ability to continue to perform tasks routinely even in the presence of unexpected events such as outages and failures.

- *Agility:* the ability to quickly and easily modify infrastructure, platform, and/or application behavior.

- *Cost-effectiveness:* the ability to reduce cost, for example by avoiding upfront capital expenditure, by eliminating managing and operating IT infrastructure costs, by consolidating applications, and by exploiting economies of scale.

Cloud Computing Enablement, Coordination and Prediction Challenges

In order for cloud computing to realize these key qualities, it is still necessary for providers, developers and end users to design several hardware and software components well – not an easy task, given the complexity of managing and using these systems. For instance, IaaS providers may offer the ability to increase or decrease the number of computational instances on demand, but elasticity can only be achieved when an application or a platform has been designed (a) to take advantage of the additional computational capacity and (b) to avoid crashing the system when resources are reduced. The stratification of the cloud ecosystem into IaaS, PaaS and SaaS providers reflects the need to offer different degrees of flexibility in influencing the final architecture, with IaaS providers allowing most of the software stack to be determined by end users and SaaS providers offering the most restrictive solution. Figure 1-2 illustrates a layered architecture of the cloud computing ecosystem.

Infrastructure layer: consists of the raw resources to be shared among users – e.g., servers, storage, network, and instruments. Upper layers implement the resource sharing mechanisms. IaaS solutions, in rare occasions, operate on this layer to provide "bare metal" resource access to end users.

Isolation layer: provides mechanisms so that multiple users can share infrastructure resources. Isolation is usually achieved with the use of virtualization technologies (e.g., Citrix Xen [18] and VMware [137] machine virtualization, and Virtual Private Network [51] and ViNe [133] network virtualization). Isolation minimizes interference (e.g., crashing of a VM does not affect the progress of other VMs running on the same resource), facilitates partitioning (e.g., multiple VMs running on the same resource, and mutually isolated virtual networks on the Internet), and allows dynamic reconfiguration (e.g., reconfiguration of machines to satisfy application requirements as in Grid5000 [21]). IaaS act on this layer to create application-tailored execution environments on demand.

Management layer: responsible for exporting high-level interfaces to monitor and control lower layer resources. For example, Amazon EC2 [7] offers services to create and destroy VM instances, with various allocation policies and pricing: on-demand instances (best-effort), reserved instances (guaranteed service) and spot instances (biding for unused capacity). IaaS providers operate on this layer for authentication, authorization, accounting, scheduling and load balancing purposes. Commercial (e.g., VMware vSphere [138]) and open-source (e.g., Nimbus [73], Eucalyptus [99], and OpenNebula [102]) software packages offer different management capabilities [122].

Appliance layer: provides execution environment images (VM images) pre-configured with platforms that can be used to host applications. Execution environments can range from

simple images containing only the operating system, to complex images containing frequently used components such as web server or relational database environments. PaaS providers act on this layer exporting the necessary platform interfaces to the development and/or deployment of applications.

Application layer: provides the actual application using the resources exported by the lower layers to carry out the necessary computation. SaaS providers operate on this layer offering a complete solution to a problem, often exported through user-friendly interfaces.

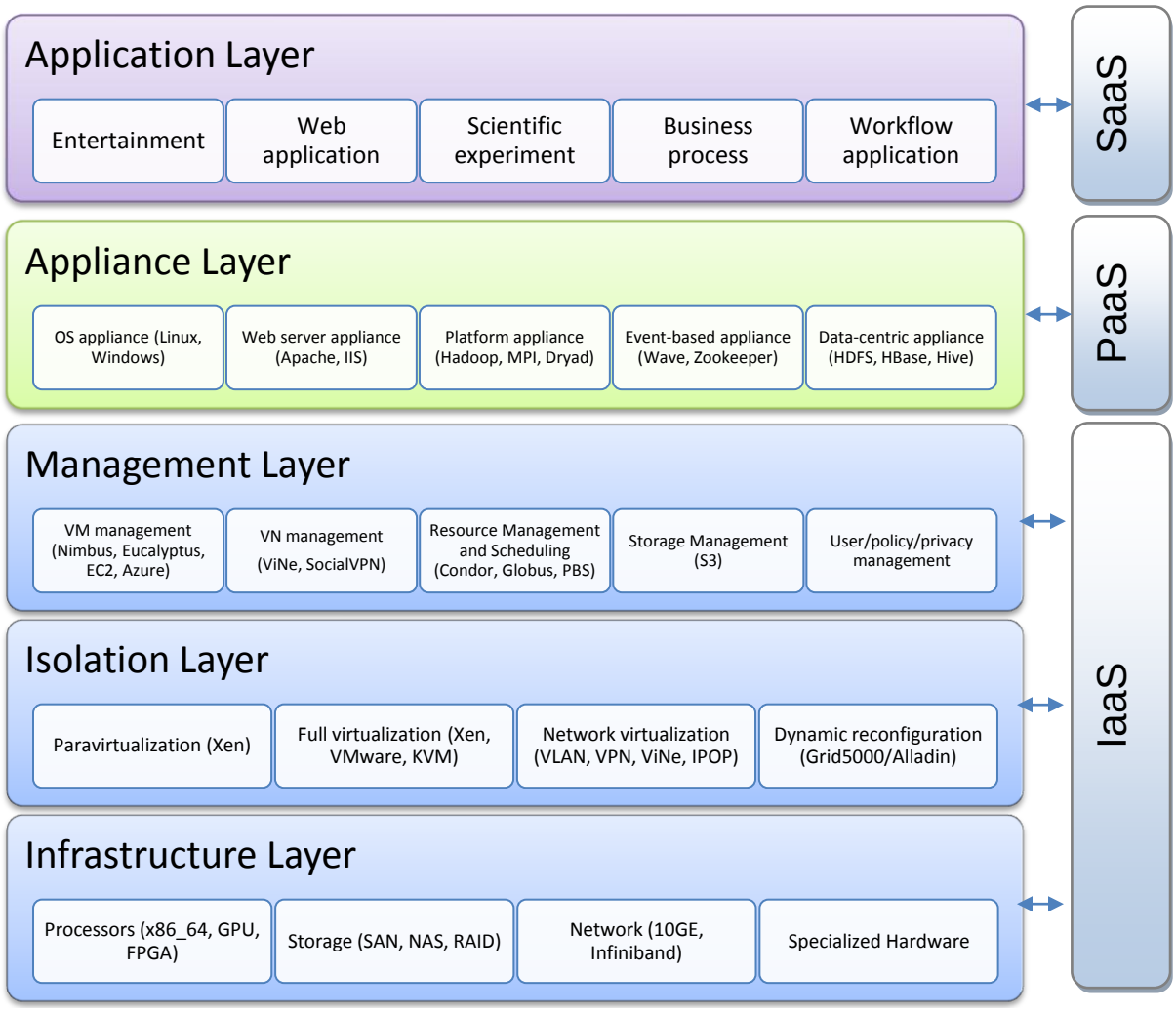


Figure 1-2. Cloud computing ecosystem. While ready to use SaaS solutions exist, many end users are required to work with PaaS and IaaS cloud providers, and deal with the complexity of choices offered in order to benefit from cloud computing.

In the context of cloud computing, the challenges faced by end users and/or developers, and addressed in this dissertation are:

- *Integration of legacy applications*: in cases where applications already exist, these need to be recast to work with modern computing systems or need to be integrated to work collaboratively with other applications. Service-Oriented Architecture (SOA) promotes standards for developing loosely-coupled interoperable services. As SOA adoption for offering SaaS grows, developers are expected to recast legacy applications as Services in an efficient manner. This problem can be tackled with tools and Integrated Development Environments (IDEs) that facilitate the redevelopment of applications as services. However, the cost of doing so can be high for two reasons: (a) some applications may have been coded in a way that it is difficult to take advantage of such tools, and (b) source code is not always available.

- *Development of distributed applications*: to take advantage of the distributed computational power on a single or on multiple clouds, development of parallel applications is required. Since each subpart of the application runs on a different resource, the application developer needs to decide how the application is to be split, how to distribute the data and the computation efficiently, and when the application subparts need to communicate in order to exchange information.

- *Management of heterogeneity*: when developing applications, heterogeneity of hardware performance and configuration as well as operating system, libraries and communication protocols offered by different IaaS or PaaS providers need to be carefully accounted for, in order to extract the best performance out of the system.

- *Application scalability and reliability*: to ensure scalability and reliability of a solution, the developer needs to investigate communication limitations, parallelization overheads, the impact of failures and the availability of management tools.

- *Application resource consumption estimation*: when accessing computational resources in a pay-as-you-go fashion, it is desirable for end users to be able to estimate the resource requirements of the applications, for example in terms of CPU, memory, storage and bandwidth required. While end users may have some knowledge about the application pattern, better and faster resource allocation and load balancing decisions can be made by IaaS providers when precise information is available.

Vision and Contributions

Instead of completely shifting the application enablement, coordination and resource usage prediction to the end user, this work envisions (Figure 1-3) SaaS providers offering easy-to-use web interfaces or service APIs of specific applications to the end user. Based on user

preferences, SaaS portals or services contact one or more data centers capable of providing computational power, storage and execution environment required by the application.

In the near future, it is expected that most large datacenters will become IaaS or PaaS cloud providers. IaaS providers offer flexibility in accepting a wide variety of execution environments through the use of machine virtualization technologies. PaaS providers allow different applications to take advantage of well configured platforms. The time or space-shared computational resources allocated for the end user on IaaS and PaaS providers are then interconnected using network virtualization technologies to enable distributed applications to be executed across multiple clouds.

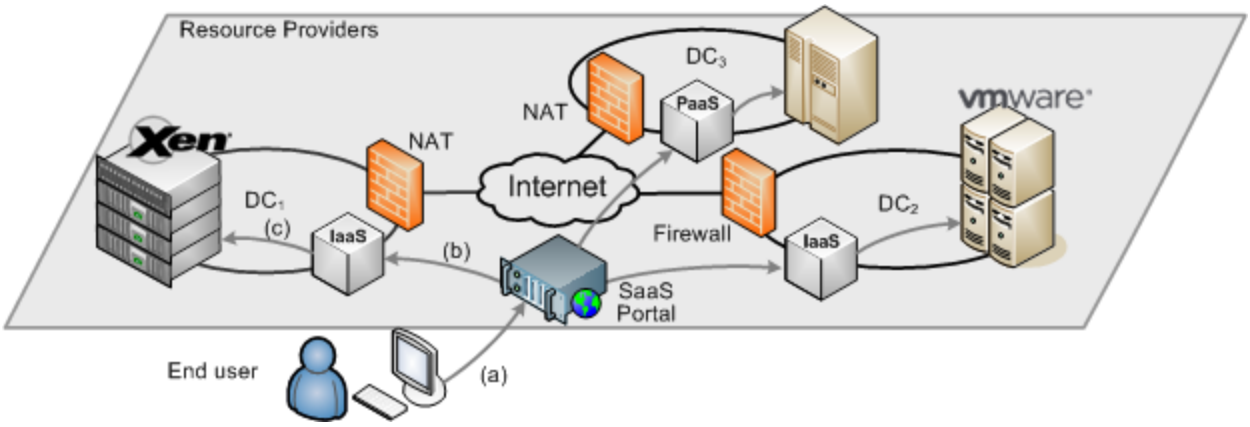


Figure 1-3. Vision on how end users will access computational resources to execute large applications. End users (a) access Web portals or services of Software-as-a-Service (SaaS) providers with an execution request. SaaS providers in turn (b) request creation of appropriate execution environment from one or more data centers (DC). DCs will offer Infrastructure-as-a-Service (IaaS) and Platform-as-Service (PaaS) solutions placing actual resources behind firewalls or NATs, leaving the responsibility of establishing the necessary connection to network virtualization technologies. IaaS and PaaS cloud providers (c) allocate and control resources according to application requirements.

This dissertation focuses on methods that address the challenges described in the previous section, and proposes tools that automate and to optimize the time-consuming operations currently carried out by end users and developers. Contributions of this dissertation fall under

three areas: application enabling and integration, coordinated deployment of applications on multiple clouds, and application resource consumption prediction. In relation to Figure 1-2, results of this work can be applied in the application, appliance and management layers, bringing benefits to all cloud abstractions: IaaS, PaaS and SaaS.

Application Enabling and Integration

To address the problem of rapidly integrating legacy applications, in particular for a subset of applications with a command-line interface, common application characteristics that need to be exposed as services were identified. Based on this classification, a methodology for enabling not only batch applications but also interactive and stateful command-line applications was proposed. In addition, Command-Line Application Wrapper Service (CLAWS), a tool that embodies the proposed methodology was developed for demonstrating the wrapping of such applications automatically as Web Services. The information about the application command-line can be provided through a user friendly web interface that allows end users to manage the service deployment and define access control.

As a proof of concept, CLAWS was successfully applied to deploy machine translation and conversational interface applications in the context of a Transnational Digital Government (TDG) [126] project. During the various development cycles of the project, CLAWS facilitated the creation, deployment and integration of Web Services in remote locations. Integrating these services into a distributed transnational information sharing system in a timely manner with minimal system downtime would not have been possible without having CLAWS enabling those applications as Web Services. To avoid acquisition of new equipment, machine virtualization technology was applied to allow the coexistence of applications with conflicting pre-requisites. Network virtualization technology was used to provide connectivity between private networks.

Coordinated Deployment of Applications on Multiple Clouds

Effectively using IaaS clouds can be challenging, especially for non-experts, given the complexity and abundance of available solutions. This dissertation presents mechanisms to enable and to coordinate the deployment of unmodified applications with embarrassingly parallel execution profiles on multiple clouds. The proposed solution scales well and can efficiently utilize distributed resources across geographically distant locations. At the same time, it does not impose high administrative overheads. The scalability is achieved by combining (a) an IaaS cloud toolkit with management services (Nimbus [53]) that allows the creation of virtual clusters with the necessary configuration on demand, (b) a virtual networking software (ViNe [133]) that recovers the all-to-all connectivity among VMs in distinct administrative domains, and (c) a MapReduce framework (Hadoop [11]) that offers parallel fault-tolerant execution of unmodified applications and that was adapted to handle pattern-based data formats and to offer skewed task distribution to deal with resource imbalance.

The Science Clouds testbed deployed at the University of Chicago (UC), University of Florida (UF) and Purdue University (PU) was used to demonstrate the scalability of a multi-cloud environment running a scientific application in the area of bioinformatics called Basic Local Alignment Search Tool (BLAST [6]). Experiments with different configurations of BLAST also showed low overhead due to machine and network virtualization, good performance of two implementations of the distributed BLAST application (one based on Hadoop and another on Message Passing Interface or MPI), and the ability of the Hadoop solution to easily accommodate new versions of the application.

Resource Consumption Prediction

Searching for the best method to automatically predict the amount of resources required by a certain job, this work surveyed and evaluated several existing machine-learning algorithms and

the influence of different selections of application-input and resource features on prediction accuracy. An extension to an existing algorithm was proposed, called PQR2, adding regression functions on the leaves of the Predicting Query Runtime (PQR) tree.

Experimental evaluation using two bioinformatics application on five different resources showed that PQR2 offered advantages in prediction accuracy over traditional algorithms (nearest neighbor, linear regression, decision table, radial basis function network, and support vector machine). Results also indicated that better predictions are accomplished when all resource and application-input features are included in the training set.

Organization

Chapter 2 addresses challenges faced in developing and deploying distributed SOA systems, focusing on the problem of turning command-line applications into Web Services in a simple and efficient manner via a Web interface. Mechanisms that automate the creation of service interfaces for specific applications minimize the development time and facilitate integration of applications, allowing fast deployment of SaaS. A transnational digital government scenario served as a case study to validate the approach.

Chapter 3 investigates the performance of combining virtualization technologies and parallel execution frameworks to run a bioinformatics application in real IaaS clouds connected via a Wide Area Network (WAN). A set of guidelines and mechanisms for extracting the full potential of the new cloud and sky computing paradigms are provided. The impact of virtualization and parallelization models on application execution performance is also investigated.

Chapter 4 surveys machine-learning algorithms and investigates their applicability for predicting job resource consumption, in particular on space-shared resources, allowing end users to make requests to PaaS and IaaS cloud providers with better estimation of cost. Then, PQR2,

the new algorithm based on Predicting Query Runtime (PQR) trees is described. In experiments using popular computationally-intensive application in the bioinformatics field, BLAST and RAxML, PQR2 outperforms traditional algorithms.

Chapter 5 summarizes the dissertation and presents future directions.

CHAPTER 2
ENABLING APPLICATIONS AS SERVICES

Many efforts are under way to implement Service-Oriented Architecture (SOA) approaches to new and existing processes. An urgent need exists for effective methods to refashion existing applications as Web Services that can be integrated into enterprise architectures that span entire organizations and/or enable cross-organization collaborations. Here, an approach to automatically wrap text-based applications into Web Services (WS) and to speed up deployment of SaaS is presented. This type of applications, often also called *command-line applications*, is common in systems and tools that process commands or data in textual form. Without the proposed approach, the integration of batch and interactive software tools would, at best, take unacceptably long times and incur large labor costs, and, in the worst case, be impossible due to missing know-how, source code or supporting infrastructure at each collaborating group.

Service Oriented Architecture

Simply stated, an SOA consists of a collection of loosely coupled services that can interoperate with each other. Interoperability is achieved by using standard languages for the description of service interfaces and the communications among services. A widely accepted technology for implementing SOAs, called Web Services [75], uses Web Services Definition Language (WSDL) and Simple Object Access Protocol (SOAP) as the standard languages for definition of, and communication among, services, respectively. More generally, WS technologies also adopt several other standards for service discovery, workflow orchestration, reliable messaging, and security. The planned widespread adoption of SOAs for digital organizations stems from the fact that, once services are known and available, processes are more easily created, maintained, integrated and reused. This, in turn, can enable agile enterprise

or government processes and facilitate new forms of interaction as well as access to functions and information.

One of the benefits of using SOA and WS is the decoupling of service interfaces from service implementations. In principle, once a service is created, it is possible to update, replace or modify its implementation without changing its interface. Keeping the interface intact is important to not affect other services that depend on it. Given an existing implementation of an application, two tasks must be performed in order to integrate the application as a service into an organization process. First, an execution environment (i.e., the hardware and other software needed to run the application) must be provided. Second, the application must be "service-enabled", i.e., additional programming is needed to "extend" the original application to behave as a service and to interact with other services. These two tasks can be particularly challenging when services based on existing applications (also called *legacy applications*) must be deployed across different organizations with possibly different IT infrastructures. The case study that provided the motivation for the work reported in this chapter is a concrete example of a scenario where these two challenges arise.

Many collaborative processes, including the case-study considered in more detail later in this chapter, require groups possibly from different countries to engage in collaborations that entail accessing and sharing information. Typical requirements include organizations being allowed to query databases of other organizations, being able to invoke translation tools to enable communication across different languages and having access to dialog-oriented interfaces to enable users to easily issue queries. These are information processing tasks for which there already exists (legacy) code. Reuse of code, in the context of SOA, can enable the quick development and deployment of effective systems. The alternative of redeveloping programs

would either be too time consuming, too expensive or both. However, in our experience, the available legacy applications run on different operating systems and require additional software that is not available in the target agencies' IT infrastructure. In addition, the legacy application code preceded the existence of WS technologies and could not interoperate in the context of a SOA. A naïve approach to overcome these problems would require the acquisition of additional hardware and software (to address the first problem), and programming skills and labor to manually modify existing applications (to solve the second problem). This naïve approach leads to large unaffordable expenses which are recurring in nature due to the unavoidable need to update, extend and modify the system implementation.

Virtualization Technology: Providing the necessary execution environment

The use of virtualization technology has been applied to address the first challenge, i.e., the problem of providing different execution environments for multiple services within organizations with distinct IT infrastructures. The key idea is to use readily available virtualization software (e.g., VMware [137], Xen [18], and Hyper-V [92]) to create virtual computers on which the software needed to run existing applications can be installed. This allows the encapsulation of conflicting execution environments in separate virtual machines, thus eliminating any incompatibilities among software environments already in place at the organizations and the execution environments needed by different applications. In conjunction with machine virtualization technologies, it is essential to deploy software that leverages network virtualization technologies (e.g., ViNe [133], VLAN [104], and VPN [51]) to enable the creation of virtual networks to connect distributed resources (virtual or physical) often placed behind firewalls and network address translation devices, while making it possible to securely create and couple distributed services. Figure 2-1 illustrates the use of VMware machine virtualization software for

encapsulating software with conflicting requirements and ViNe network virtualization software to provide connectivity in the TDG case study, described in more detail later in the chapter.

While machine and network virtualization address conflicting requirements at the hardware, software and networking levels, Services and SOA can be used to address software interoperability problems. Towards this goal, the focus of this chapter is on the second challenge - how to enable applications as Services in order to allow them to interoperate with other Services.

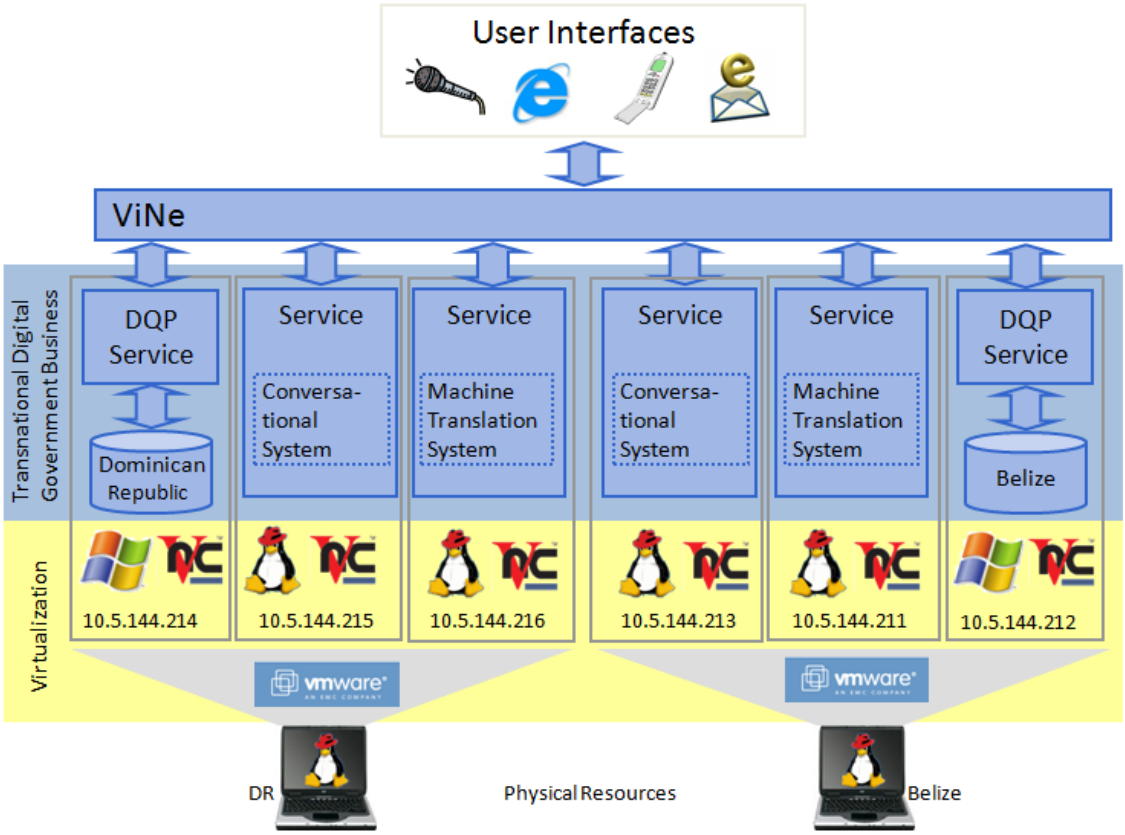


Figure 2-1. Providing execution environment and connectivity with virtualization technologies. VMware creates different execution environments in the same physical resource, allowing TDG components with conflicting requirements to coexist. ViNe offers connectivity recovery, transparently connecting resources in multiple private networks. While developers operate individual VMs through VNC, end users interact with TDG services and web interfaces through browsers and receive notification messages via e-mail and mobile phone.

Wrapping Applications as Web Services

At a very basic level, there are some key requirements for a Service to run as such. First, a container is needed, i.e., Services require software that receives and processes service requests, executes the corresponding program, and returns the application response to the client. In other words, a container provides a runtime environment for Services written in a particular language (e.g., Java) and offers mechanisms to manage the deployment and use of multiple Services. Second, for a service to be useful, code is needed to invoke the Service (the client side) as well as to provide the Service (the server side). In theory, the code needed to implement a Service should work independently of the container used. However, in practice, Web Service developers must be very knowledgeable and careful about the containers and languages to be used. If done manually, enabling a complex legacy application as a Service can be a time-consuming and expensive process. This is particularly true since WS technologies are still relatively new, lack sophisticated programming environments, and require non-specialized programmers to learn new skills and tools. When multiple organizations with independently-administered IT infrastructures are involved, the expertise and effort needed to turn applications into Services increases in proportion to the number of organizations, due to potential use of different containers and languages. While the virtualization approach discussed earlier can be used to alleviate this problem, it needs to be complemented by tools to automatically enable applications as Web Services.

For the above-mentioned reasons, the automatic deployment of Services is an active area of research and development in academia and industry. GAP [112], GFac [71], SoapLab [113], and VAS [84] are examples of approaches proposed and developed for scientific applications to be executed in Grid computing environments. These Grid-oriented solutions lack important features, namely:

- *Easy wrapping of text-oriented applications into WS:* although existing enabling tools hide many of the complex aspects of WS deployment, it is still challenging for users to have to deal with different text input sources and formats and to provide the infrastructure required by the enabling tools themselves.

- *Support for interactive applications:* existing solutions, in general, only support batch applications, i.e., applications that do not require interaction with the user while executing the application.

- *Simple mechanisms for authentication and authorization:* the goal is to avoid using complex security solutions not yet available across all vendors and platforms such as Grid Security Infrastructure (GSI) [55] and WS-Security [100] when other commonly used techniques are sufficient (e.g., hypertext transfer protocol basic authentication combined with transport layer security).

- *Platform independence:* existing tools target Linux environments on x86 machines or other specific execution platform, and cannot completely or easily handle applications developed for other platforms.

While the approach presented in this chapter is generally applicable, it is motivated by the need to integrate machine translation software, conversational interfaces and a database query system into an SOA for TDG to enable collaboration among government agencies in different countries. The independently-developed applications require different execution environments and present the described system integration challenges.

Web Service Development and Deployment

A Web Services *development environment*, similar to a typical software development environment, makes use of existing programming languages and compilers. The key difference is the need to interface with specific libraries or artifacts created by a WS engine vendor (see below). IDEs and WS tools facilitate the development of Web Services by exposing WS APIs, providing intuitive graphical user interfaces, generating skeleton code, and automating the deployment and testing phases of a development cycle. Figure 2-2 illustrates the software components stack of the WS deployment environment (also called runtime), where Web Services actually run. The stack includes the following:

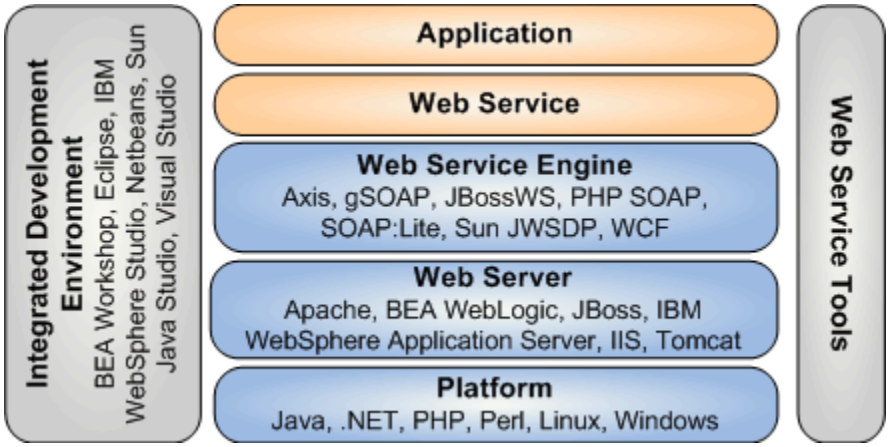


Figure 2-2. Typical runtime environment for a Web Service. Composed of a platform where a web server hosts a WS engine. The WS engine, in turn, makes an application available as a Web Service. IDEs and WS tools facilitate the development and deployment of Web Services.

- *Application*: software that implements the functionality to be provided (e.g., a calculator, a translator, a query processor, and a natural language parser). Applications may already exist or can be developed anew.

- *Web Service*: component that receives requests from the Web Service engine and invokes the appropriate application. Once the application executes, results are transmitted back to the engine. For applications developed in languages such as Java and C#, this could simply be annotations added to the existing application source code. In scenarios with applications developed in other languages or when source code is not available, development of code is required as detailed soon after this stack description.

- *Web-Service engine*: Web Services container, responsible for exposing a standard and interoperable interface, and for offering Service management mechanisms (e.g., Apache Axis2 [10], gSOAP [40], PHP SOAP [105], SOAP::Lite [120], Sun Java Web Services Developer Pack (WSDP) 2.0 [128], and Windows Communication Foundation [90]). Commonly implemented as web applications, WS engines process received SOAP messages and forward the information formatted according to the Service definition to the appropriate WS. WS engines also encapsulate responses from the service application into SOAP messages, before transmitting them back to the requesting client. In general, engine vendors provide WS tools to manage the deployment of WS, and IDEs make use of these tools targeting a specific Web server.

- *Web server*: Web application container that implements Hypertext Transfer Protocol (HTTP) for interactions with clients (e.g., Apache HTTP Server [12], Oracle WebLogic [103], IBM WebSphere Application Server [63], Microsoft IIS [91], JBoss [106], and Tomcat [13]). Sophisticated web servers tend to offer additional capabilities (e.g., authentication, load-balancing, and logging), and to support an increasing number of web application types (e.g., applications backed by databases).

- *Platform*: runtime environment of Web server, WS engine, Web Service and application (e.g., Java VM, .NET CLR, and Linux on a x86-based computer).

As previously mentioned, Web Services are defined using the Web Service Definition Language (WSDL). There are two main approaches to develop Web Services:

- *WSDL-to-code*: From the WSDL description of the service interface, this approach generates the skeleton code for both the server and the client sides of the Service. Then, the developer is required to fill the skeleton with code that implements the actual functionality of the service.

- *Code-to-WSDL*: This approach starts from existing code and then derives the WSDL for the selected Application Programming Interface (API) to be exposed as Service. In this context, the "existing code" could be one of the following: the source code of an existing application; binaries of applications written in a language that offers reflection (e.g., Java and C#); or the skeleton source code of the API in the case of applications to be developed anew.

WS developers recommend the first process, as the likelihood of Services working across different implementations of WS engines and Web servers is highest when the WSDL interface is defined following the standard specifications, in particular, meeting the WS-I Basic Profile 1.1 [17] (other recent versions of the standard are in progress). When exposing existing applications as Services, often times it is easier to start from the source code and automatically generate the WSDL files using tools offered by the WS development environment. However, this automated process may still lead to interoperability problems due to the generation of WSDL constructs that are not fully supported in different environments. This is evident from the fact that WS engines from different vendors are not fully interoperable, even when Services are developed using the same programming language (e.g., Axis and Sun Java WSDP). In spite of the existence of tools that facilitate the creation of Services, in many occasions, the development of Services is still a hard task for the following reasons:

- WS development tools known to a developer may not support the language used to develop the application to be enabled as a Service.

- Required modifications in the source code of applications can be too complex for programmers who are unfamiliar with the application or are not advanced WS programmers.
- The source code of the application may not be available.
- Automatically generated code may not be complete, and specialized knowledge is required to add the missing code and/or fine-tune the resulting Web Service code.

Despite the fact that recent versions of IDEs support the development of Web Services, in general, IDEs tend to concentrate on a specific version of WS technologies and target a specific Web server container. For instance, Visual Studio 2008 [88] IDE supports .NET Framework 3.5 and targets the Internet Information Services (IIS) container. IDEs facilitate not only the development but also the deployment and test of the Services. Still, IDEs and integration tools do not solve the difficulties pointed above or may not offer facilities for the developer in certain environments. For instance, although Netbeans 5.5 [98] allows Sun Java WSDP to be deployed on either Tomcat or Sun Application Server containers, testing features available for each container differ greatly. While Netbeans 5.5 IDE enables the developer to test a Web Service deployed to a Sun Application Server from the IDE itself without the need to create new code, the same is not true when the target container is Tomcat. Similarly, support for Axis2 WS engine became a reality only after Netbeans version 6.1.

Commercial and open source systems that support Enterprise Application Integration (EAI) (e.g., Oracle WebLogic, IBM WebSphere, and JBoss) focus on modernizing and consolidating applications in an enterprise, exposing them not only as WS but possibly as Remote Procedure Call (RPC), Remote Method Invocation (RMI), Message-Oriented Middleware (MOM), and Java Connector Architecture (JCA) among others (see [66] for an overview of these technologies). When dealing with legacy applications, complex tasks of configuration and programming of adapter components for each legacy application are required.

Once again, the cost of EAI systems, the necessary support from the vendors, and large labor costs might not be affordable. When using SOA for EAI, exposing existing applications as Services faces the problems discussed above. In this context, vendors provide ready-to-use adapters only for known legacy protocols (e.g., Information Management System, Customer Information Control System, enabling COBOL and PL/I applications). This chapter proposes a solution that complements SOA-based EAI by simplifying the process of WS-enabling text-based legacy applications independently of the programming language used and without requiring the existence of data access mechanisms.

The conclusion from this section's brief review of the challenges faced in the development and deployment of Web Services is that developers still need to find workarounds that address potential interoperability problems. When possible, developers can tweak the WS technologies, add features to the existing tools, and propose changes to be incorporated by the vendor, but these are time-consuming steps that are not for the faint of heart.

The solution proposed targets text-based applications, including interactive ones (specific characteristics of these applications are discussed in the next section). It does not require source code to be available, achieving programming language independence and avoiding the need to modify existing applications. An important advantage of the proposed solution is that it enables IT staff to keep Services operational with minimal or no knowledge about the intricacies of a particular WS engine or WS standards, even when applications are updated.

Application Classification

The approach and coding needed to wrap an application as a Service depends on the application's characteristics, in particular on how it behaves at runtime. This section differentiates several types of applications in order to identify the features that an application wrapper service (proposed in the next section) needs to provide. Applications can be invoked via

a Graphical User Interface (GUI), a text-based interface (command-line specification, text files, and standard input and output) or an application programming interface (API). The solution presented in this chapter focuses on text-based applications, which can be further classified as shown in Table 2-1.

Table 2-1. Text-based application classification.

Non-interactive (or batch)	Applications that once started, run without any user interaction. Input parameters are specified when starting the application and output is collected at the end of execution.
Interactive	Applications in which users enter individual commands and control the execution flow. Typically, new commands are accepted only after the processing the previous command finishes and results are returned.
Stateless	Applications in which the processing of a command does not depend on data computed by the application in response to past commands.
Stateful	Applications in which the outcome of a command depends on data computed by the application in response to past commands.

Well-known commands used in Operating Systems (OS) can be used as examples that illustrate the types defined in Table 2-1. Non-interactive applications are necessarily stateless (e.g., echo, ls, and dir), while interactive applications may be stateless (e.g., cat, and copy files) or stateful (e.g., sh, and matlab). From a WS perspective, it is important to understand the programmatic means of accessing an application at runtime. Text-based applications are not invoked directly by the clients. They are invoked by WS engines once client requests are processed. Particular care needs to be exercised when turning a text-based application into a Web Service, since the Service invokes and interacts with an application that is executed outside of the WS container. Further, for interactive applications, it is important for the Web Service to identify its users and decide how to maintain application state(s). In some scenarios, it is desirable for the application state to be shared among all Service users (in which case the Service invokes and interacts with one instance of the application, ordering requests to avoid conflicts). In other scenarios, the application state must be independently maintained per user or session

identification (in which case the Service instantiates one application per user or per session identification).

Depending on how applications are invoked, more than one type of usage mode may be available. For example, the Machine Translation (MT) system used in the TDG project (further details later in this chapter) provides two executable programs: one that translates from English to Spanish (eng2spa) and another from Spanish to English (spa2eng). It accepts the text to be translated as input (acting as batch stateless application) or it can expose a console-based user interface, in which case it translates multiple sentences following user's input (acting as interactive stateless application). The communicator system (CS), also used in the TDG project is an example of interactive stateful application. CS exposes a console-based user interface where users can type queries in natural language (e.g., 'who entered Belize in December 2006?'). From the user-typed queries, CS generates Structured Query Language (SQL) queries, sending them to the actual databases. Since CS keeps the conversation state, a subsequent query could be 'list the passengers coming from USA' (among those who entered Belize in December 2006).

The Unified Modeling Language (UML) sequence diagrams in Figure 2-3 provide the sequences of interactions that occur between a Web Service client and a Web Service for the three types of application identified above. One instance of batch stateless application must be created in order to handle each request from Web Service clients (Figure 2-3a). Interactive applications (Figure 2-3b, and Figure 2-3c) are instantiated and reused over several requests. In the case of interactive stateless applications, one or more instances of applications can be created, and requests, independently of the requesting Service client, can be directed to any application instance, opening opportunities for load balancing. In Figure 2-3b, only one application instance handles all requests, forcing the Service to queue requests and to allow only

one request to proceed at a time. Handling interactive stateful applications require the Web Service to identify the requesting Service client, and to direct requests to the appropriate application instance that keeps state for the corresponding client.

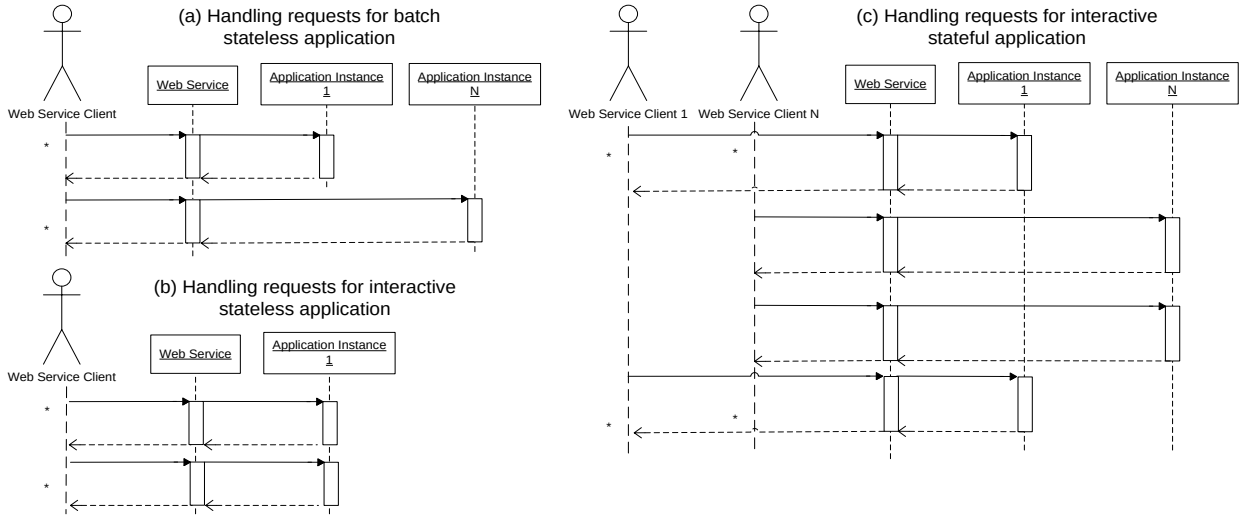


Figure 2-3. Sequence diagrams depicting the necessary interactions between WS client, Web Service, and application for different types of application. For batch stateless applications (a), one application instance is created for each request whereas for interactive stateless applications (b), an application is instantiated once and re-used over several requests for multiple clients. However, for interactive stateful applications (c), an application instance is created and shared among requests from a specific user or a group of users, according to the access control mechanism chosen when WS-enabling the application.

Figure 2-3c shows that requests from the Service client 1 are always directed to application instance 1, whereas requests from Service client N are always directed to application instance N. In this case, a Service client can be a single user or a group of users.

Command-Line Application Wrapper Service

The tool developed in this work, called Command-Line Application Wrapper Service (CLAWS), focuses on interactive text-based applications, but also supports batch applications. It automatically wraps, builds and deploys an application onto a web-server container hosting a particular WS engine (Axis). CLAWS enables its users (in general, IT staff) to easily generate a

Web Service for existing interactive text-based applications through a web form without expecting the user to have knowledge about the various WS standards or to write additional code.

The information that needs to be provided to CLAWS to initiate the automated process consists of a

- *Service Name*: name of the Service to be generated;

and, for each operation to be provided by the Service, a data set composed of the following:

- *Operation*: name of the Service operation, along with the name of input attributes for the Service operation; the input attribute name can be referenced when specifying the command-line, the environment variables or the standard inputs (see below). Support for arrays of inputs is also available, enabling the creation of parameter-sweep applications, i.e., applications that are executed for a range of parameter values instead of a single value.
- *Command-line*: entire command-line exactly as invoked by the deployed Web Service.
- *Environment variables*: set of environment variables to be set before the execution of the application.
- *Current working directory*: indicates the directory from where the application will be invoked.
- *Standard Input*: information sent to the standard input of the command-line application process.
- *Interaction end-detector pattern*: interactive applications accept inputs from standard input and output results. Since the Service proxies the interaction between the Service client and the application, it needs to detect, based on the output of the application, when the processing of an input has finished. The end of an interaction is in general detected when a specific pattern is output (e.g., a prompt). This parameter indicates how the Service can detect the end of an interaction.
- *Rewrite rules*: pairs consisting of a pattern and a value. If the pattern matches parts of the output, each part is replaced by the corresponding specified value. Rewrite rules are especially useful for removing a prompt (part of the output of the application) from the Service response.
- *Mode*: indicates the mode of operation of an application. It can be batch, stateless interactive or stateful interactive (Figure 2-3). When an application is configured as

interactive, the end user needs to provide the waiting time, in the form of number of tries to match the end pattern with the interval between attempts. The deployed Service will monitor the output of the application and will try to match the end pattern the specified number of times. Note that a stateless interactive application could be also executed as non-interactive. However, this is not recommended for applications with high initialization overhead, as each service request requires unnecessary effort to initialize the application.

- *Security configuration*: a Service may require access control, in which case user authentication is necessary. Secure and well established HTTP basic authentication combined with Transport Layer Security (TLS) is currently supported by CLAWS. Supported access control modes are as follows: no authorization scheme, user-based authorization, or role-based authorization.

The above-listed information is sufficient for the generated Service to offer exactly the same capabilities as the original application. The user of CLAWS can also choose to input different information so that the Service generated by CLAWS has: a reduced set of capabilities (by not exposing options that the original application accepts), an increased set of capabilities (e.g., offering a parameter-sweep capability), or a combination of capabilities from several applications (e.g., inputs could determine which of several different programs would be invoked).

The steps taken by CLAWS to automatically wrap and deploy applications as a service are:

- *Code generation*: generates the implementation of the Web Service responsible for instantiating the application according to the information given by the CLAWS user. The generated code assembles the command-line as specified, invokes the application according the mode of operation specified, handles inputs and outputs and provides authorization when required.
- *Deployment descriptor generation*: specific deployment descriptors are generated depending on the targeted WS engine to instruct the engine on how to deploy the Service.
- *Building*: the generated code is compiled before the Service can be deployed.
- *Deployment*: the compiled code is deployed on the targeted Web Service engine.

To exemplify the CLAWS wrapping process, consider the commonly used OS command ‘cat’, an application that concatenates files and prints them on the standard output. Suppose that the goal is to deploy a Service with two operations: one that simply copies the input data to

output and another that stores the input to a file serving as a clipboard. Figure 2-4, shows the information needed to deploy a Service called ‘MyService’ with an operation ‘echo’ available for users ‘user1’ and ‘user2’ and an operation ‘store’ available for users with role ‘users’.

Service Name:	MyService
Operation 1:	echo message
Command 1:	cat
Input 1:	<message><end of interaction>\n
End pattern 1:	(?s).*<end of interaction>\n$
Rewrite rule 1:	<end of interaction>\n$ []
Mode 1:	Interactive and stateless
Security configuration 1:	User-based user1 user2
Operation 2:	store text
Command 2:	cat > C:\temp\clipboard
Input 2:	<text>
Mode 2:	Non-interactive and stateless
Security configuration 2:	Role-based users

Figure 2-4. Example of input to CLAWS to wrap a text-based application. The Web Service, called MyService, offers two operations: echoing a message and storing a text. Both Service operations, when deployed, invoke the common OS application ‘cat’. The first operation specifies ‘cat’ as an interactive stateless application, adding a pattern to distinguish different interactions. The second operation specifies ‘cat’ as a non-interactive application and stores the text provided by the client into a file defined by the application enabler.

For both operations, the new Service invokes the ‘cat’ command. The first operation expects the ‘cat’ application to be used interactively. Each request from clients will specify different input messages. Messages are passed to standard input of the application with an extra string to facilitate the detection of the end of an operation. The example in Figure 2-4 – ‘(?s).*<end of interaction>\n$’ – illustrates the support for Java style regular expression for additional flexibility. The second operation executes ‘cat’ in non-interactive mode, and expects

the text to be stored (parameter ‘text’) as input parameter. The text input is sent to the application’s standard input, and ‘cat’ stores it into a file pre-defined by the application enabler as ‘C:\temp\clipboard’. Since ‘cat’ does not require environment variables to be set and it can be executed from any location, the current working directory and environment variables have been omitted in the figure.

The first operation ‘echo’ can also be configured as non-interactive without an end pattern and still perform the same desired functionality. This illustrates the flexibility offered by CLAWS, supporting applications in both interactive and non-interactive modes. Furthermore, a tool like CLAWS can be configured to create a pool of stateless interactive application processes to serve simultaneous requests from different clients without serializing the requests in order to provide better response performance (as shown in Figure 2-3b).

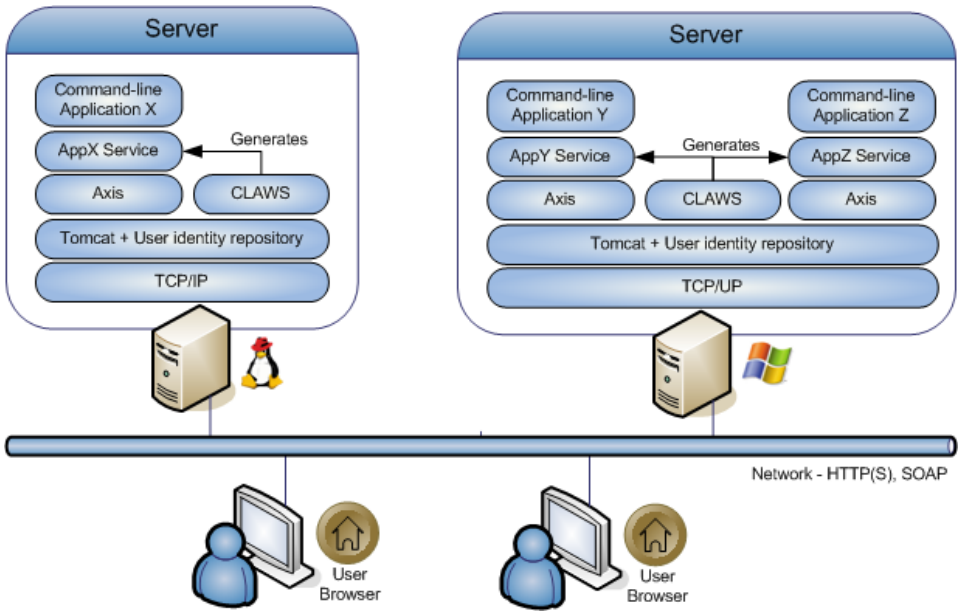


Figure 2-5. Enabling applications as Web Services with CLAWS framework. A web interface is offered for the user to enter information about the application to be enabled as Web Service. Once the information is validated, the Service is generated, built and deployed in the same container as CLAWS, independently of the execution environment, becoming available for use by other applications on the network.

Figure 2-5 depicts how CLAWS, developed using Java 1.5, is deployed as a web application on Tomcat. Using JSP and servlets technology, CLAWS delivers a web form similar to Figure 2-4 to its users. Once inputs from users are received, CLAWS generates and deploys Axis Web Services 1.4 into a Tomcat Web server container. This combination of technologies was chosen for offering easy deployment in most platforms. In the TDG project, CLAWS was deployed in both Linux and Windows environments. If other combinations of programming languages, web servers and WS engines are required, CLAWS concept can be applied to the particular environment. Note that although developed in Java, CLAWS can wrap text-based applications developed in any existing language without requiring their source code or the manual development of new code. Testing of the newly created Service can also be performed through a web user interface.

From the security perspective, CLAWS makes use of the commonly used basic HTTP authentication-based access control mechanism. This is sufficiently secure, when combined with TLS, in many scenarios, including those faced in the TDG project. More complex solutions (e.g., security based on delegation and on standard X.509 certificates as defined by the Globus project [55]) could also be implemented. However, CLAWS does not support these solutions since security standards in WS are still evolving and not fully supported by many stable WS engines. Another aspect to consider when creating new services is the validation of service inputs. Services enabled by CLAWS assemble a command-line to be executed, including application executable and input parameters, based on the service operation inputs. Manipulation of operation inputs can potentially result in exposing the system to non-authorized activities such as access to files, spawning of malicious processes, and access to data through SQL injection. Current implementation of CLAWS assumes that applications wrapped as services provide

sufficient mechanisms to validate inputs to prevent undesired activities. Nevertheless, many applications would benefit from input validation if offered by CLAWS. A language for specifying and validating complex application command-lines was proposed in [149] with support for input inter-dependencies and input type. Providing a comprehensive set of validation mechanisms through simple interfaces requires further investigation and is subject of future work.

Transnational Digital Government Case Study

The TDG project [126], an NSF-funded research aimed at deploying an information system that enables the sharing of immigration information among border-control agencies that collect and use such information in Belize and the Dominican Republic. The requirements of such a system included the ability to transparently query databases in different countries, the automatic translation of information expressed in a language that differs from the native language of the user, the use of conversational interfaces to facilitate queries to be issued by border agents in natural language, the ability to construct and maintain a global database schema that reconciles the different local database schemas, and the configuration of events that trigger the notifications to be sent to specific agents.

For economic and other practical reasons, the information system had to be implemented without requiring modifications of the IT infrastructure of the national immigration agencies. In addition, the availability of software already developed by the project leaders for machine translation and conversational interfaces lead to the decision to reuse these legacy applications as the basis for the corresponding capabilities of the TDG system. The other components were developed anew. By design, the TDG system was conceived as an SOA whose key components were to be implemented as Services.

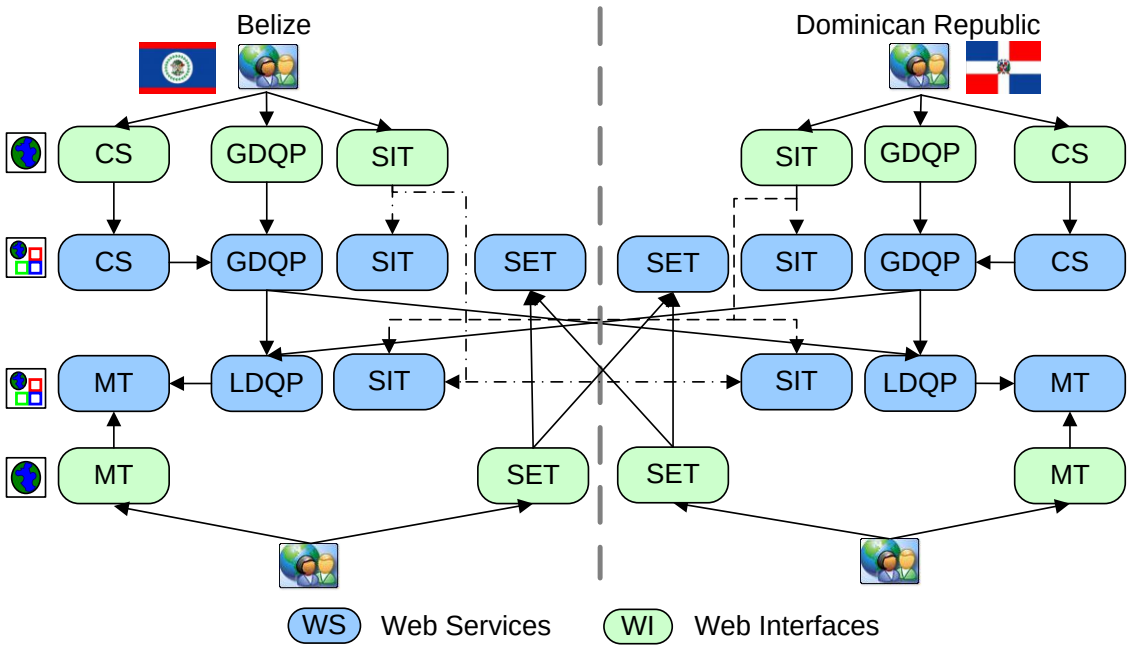


Figure 2-6. Web Services and Web interfaces built for the TDG information sharing project. An immigration agent in Belize issues questions in English using the communicator system (CS) Web interface, which in turn connects to CS Service to generate a global query to the global distributed query processor (GDQP) Service that splits the request to local distributed query processor (LDQP) Services in Belize and the Dominican Republic; the results are translated from Spanish to English (if necessary) using machine translator (MT) Services and returned to the officer.

More specifically, the TDG system components that needed to be integrated as Web Services included: an example-based Machine Translator (MT) system [24], which translates natural language texts from Spanish to English and from English to Spanish; a Communicator System (CS) [140] capable of maintaining a textual conversation with a user typing queries in natural language to the system; and a Distributed Query Processor (DQP) system [126], which integrates information from databases with distinct schemas. DQP consists of the following subcomponents: a Global Distributed Query Processor (GDQP), a Local Distributed Query Processor (LDQP), a Schema Export Tool (SET) and a Schema Integration Tool (SIT). SETs deployed in each country export the local schemas to all collaborating countries. Once a global schema is defined as mappings between heterogeneous local schemas, SIT distributes the global

schema for use by all GDQPs and LDQPs. Database queries received by GDQPs are distributed to participating LDQPs, which perform the actual search on local databases. Figure 2-6 shows each of these components wrapped as a Web Service in order to run in the computing environment required by each application and still be accessible to other components. End users in Belize and in the Dominican Republic interact with these services through user-friendly Web interfaces integrated in to a portal personalized to each country.

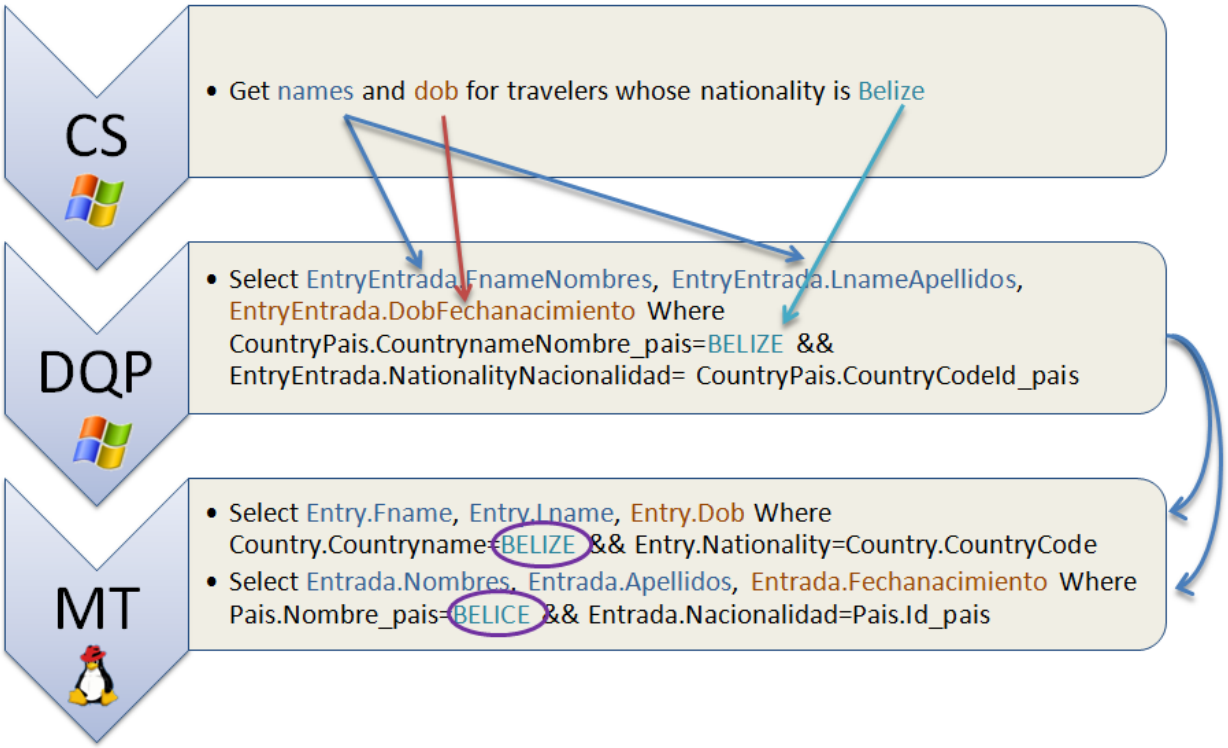


Figure 2-7. Steps during a typical use of the integrated TDG system. The CS service receives a question from an immigration agent in English. The question is transformed into an SQL query using global attributes and preserving attribute data (top arrows). The global query received by the DQP service is split and converted into multiple local queries using local attributes (arrows on the right). Whenever a field requires translation of its contents (the country name in this example), the MT service is invoked to translate the data (circled). For simplicity, the SQL 'from' field was omitted in the example.

Figure 2-7 exemplifies the steps of a typical use of the integrated TDG system by an immigration agent. The question "Get names and dob for travelers whose nationality is Belize,"

received by the CS service, is converted into an SQL query that uses attributes defined by the global schema (e.g., dob, date of birth, is transformed into 'EntryEntrada.DobFechanacimiento') and passed to the DQP service. DQP splits the global SQL into queries that use the local schema attributes (e.g., 'EntryEntrada.DobFechanacimiento*'* becomes 'Entry.Dob*'* to query the Belize database and 'Entrada.Fechanacimiento*'* to query the DR database). When attributes contain data that require translation, DQP sends the data to MT service (e.g., to translate 'Belize' into 'Belice' before querying the DR database).

The design of CLAWS (and the need for such a tool) is a direct result of the experience developing the TDG system through its two versions. In the first version, all Services were manually created, a time-consuming process that required exchange of information among research and development groups at different locations with hard-to-coordinate schedules. Since MT and CS were programmed using the C language and developed for execution on Linux-based platforms, the Services originally created to wrap these applications were developed for such an environment. In a second version of the TDG system, since Windows systems were the dominant system at immigration agencies, an attempt was made to create and support MT and CS binaries for Windows. While CS was successfully ported to Windows, such port was not practical for MT as several incompatibilities and dependencies on libraries were detected – it was decided to instead concentrate the team's efforts on improving translation accuracy. During this process, Service wrappers had to be modified to support particularities of Windows for both CS and MT, as partial implementations of the MT became available. Moreover, during the course of this project, WS interfaces and the WS client library had to be maintained manually as new version of the applications became available. The labor-intensive redesign and maintenance activity with constantly-changing versions of components is only possible if a WS specialist is available. This

increases development cost and is not realistic at deployment sites. Once developed, CLAWS greatly enhanced the team's ability to cope with and integrate these modifications (e.g., a new application option, a new command name or location, removal or addition of an error message) into the TDG system, making it possible to quickly update and continuously maintain the system available for testing by the team members.

To exemplify the use of CLAWS in the TDG project, Figure 2-8 presents the information that was necessary to enable the MT application and Figure 2-9 shows a screenshot of the CLAWS Web interface when enabling the Windows version of CS application.

Service Name:	Translator
Operation 1:	translate from to message genre
Command 1:	<from>2<to> -q
Input 1:	:genre <genre> <message>\n
End pattern 1:	(?s).*\n$
Mode 1:	Interactive and stateless
Operation 2:	translate from to messages[] genre
Command 2:	<from>2<to> -q
Input 2:	:genre <genre> <messages>\n
End pattern 2:	(?s).*\n$
Mode 2:	Interactive and stateless

Figure 2-8. Wrapping Machine Translator (MT) command-line as Web Service. The inputs show a request to create a *Translator* WS with two *translate* operations that differ only in the number of messages that can be translated in a single call. The inputs *from* and *to* form the command to be issued, while the inputs *genre* and *message* are provided as standard input. To avoid time-consuming initialization of MT the application is specified as interactive.

Basically, there were two command-line applications for MT to be deployed in Belize and Dominican Republic servers: eng2spa, and spa2eng (which provides translation from English to Spanish and from Spanish to English respectively). These commands are formed joining the

‘from’ and ‘to’ input parameters of the Service. The result of the translation can be obtained passing the ‘message’ parameter as input to the invoked application. As pointed out before, MT is a stateless application that could be executed in a non-interactive way. However, as MT requires the load into memory of a large database containing example sentences that fine-tune the translations for a specific domain, initialization of the application takes more time than a single request for translation. Thus, MT performs best if implemented as an interactive application. MT Service is also an example that integrates more than one interactive text-based application into a single Service.

Different than MT, the CS application keeps track of user requests in order to establish conversational context. Therefore, it is required to be configured as a stateful interactive application. The prompt ‘==>’ works as an interaction end-detector, and the Service input ‘text’ is passed as standard input. Furthermore, CS modes of operation are controlled through environment variables (‘CI_LAN’ and ‘CI_XMLOUTPUT’), unnecessary text from the application output are eliminated with rewrite rules, and only authenticated CS Service clients belonging to defined roles are authorized to proceed. All the described configurations are depicted in Figure 2-9.

Summary

The net result of having CLAWS available is that Services exactly with the same characteristics of manually created ones can now be automatically generated, built and deployed. WS complexities are completely hidden from the enabler, and software component updates are simplified, providing an agile solution to the integration of complex systems. The main contributions of the work are:

- Review of existing WS development and deployment environments.

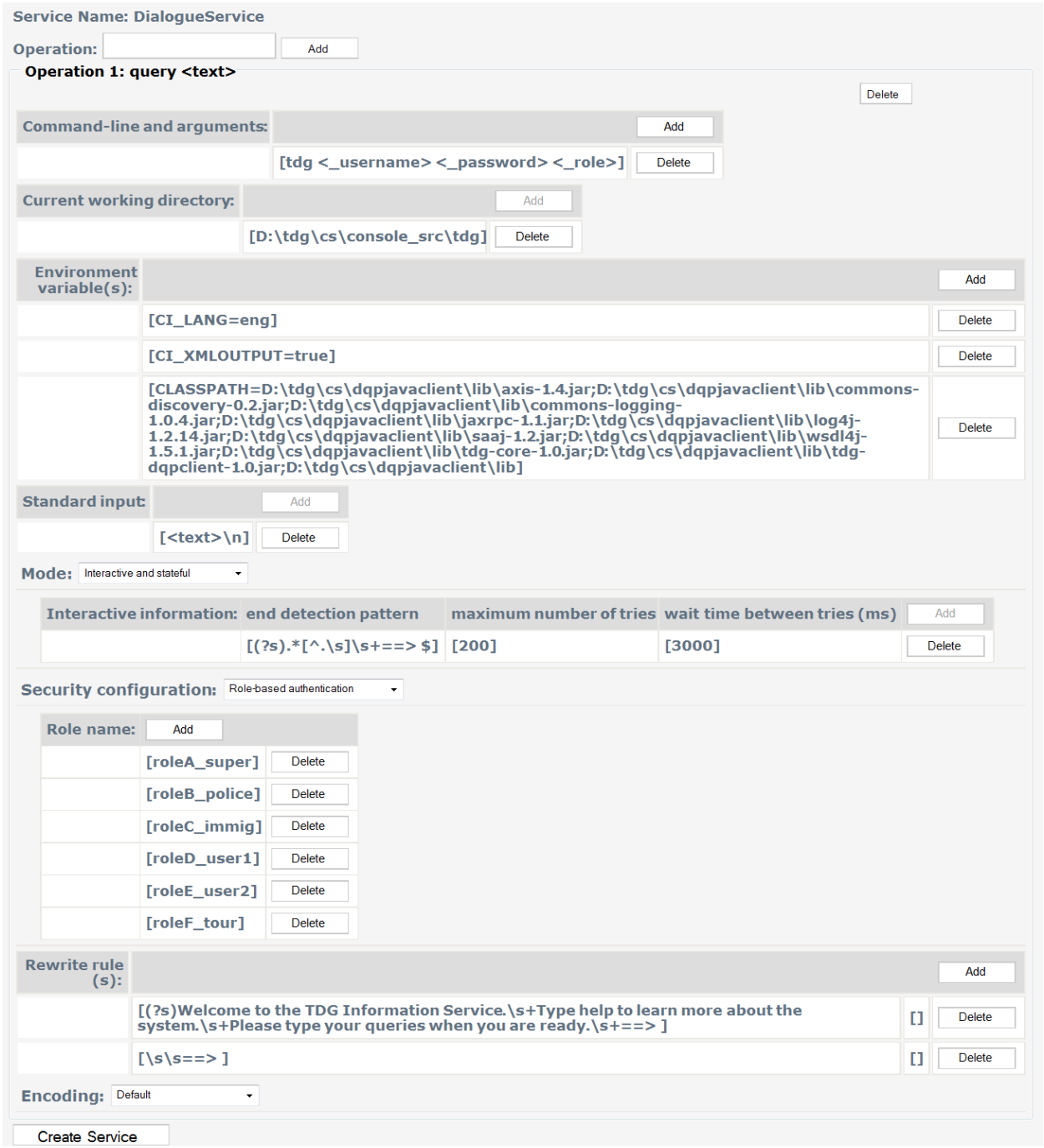


Figure 2-9. Wrapping Communicator System (CS) command-line as a Web Service. The CLAWS web interface screenshot shows inputs to request the creation of a *DialogueService* WS with a *query* operation. The *text* operation input is entered as standard input when issuing the command *tdg*. CS requires several environment variables to be set for controlling the behavior of the application and for acting as a DQP service client. Rewrite rules remove unnecessary text from the response before returning it to the user.

- Classification of text-based applications, creating a systematic approach for enabling such applications as Services.

- Development of a tool called CLAWS. Implemented in Java and deployed on Tomcat with Axis, CLAWS provides a framework for rapidly wrapping text-oriented applications as Web Services. The approach addresses the problem of efficiently integrating legacy applications without modifying its source code. CLAWS targets interactive and stateful applications, which are common in digital government domains and not supported by existing solutions in grid systems.

- An evaluation of the effectiveness of the proposed approach and CLAWS in the context of the Transnational Digital Government Project. While the initial version of the TDG system required hours or even days to implement and test Services manually, the CLAWS-enabled Services were deployed and tested within minutes.

This chapter presented a methodology for facilitating the enablement of unmodified applications as Services, assuming that resources supporting the applications of interest are available in multiple sites. The direct benefit of this approach is that end users can combine applications to work collaboratively as a single complex system. The next chapter looks into how to prepare and make use of a distributed execution environment using emerging technologies.

CHAPTER 3
COMBINING SKY COMPUTING, VIRTUALIZATION AND MAPREDUCE

Researchers in many fields of science need to routinely conduct large-scale computations on distributed resources. In this scenario, end users can benefit from the tremendous computational capacity offered by cloud providers when taking advantage of the following emerging technologies:

- *Distributed middleware* for connecting data/cluster computing centers; this includes Grid-computing middleware [29][48][52][101] for user authentication and accounting, remote job submission, resource scheduling/reservation, and data management;
- *Virtualization technologies* capable of providing application-specific execution environments; these include hypervisor-based virtual machines (e.g., Citrix Xen [18], VMware [137], Microsoft Hyper-V [92], and KVM [78]), virtual networks (VNs) [51][69][104][129][133], VM and VN management middleware [73][99][102][107];
- *Programming platforms and runtime environments* that facilitate the parallel execution of applications as concurrent jobs on subsets of the input data (bioinformatics is an example of a scientific domain where many such applications exist.) These environments greatly facilitate or transparently provide management of application execution, e.g., for scheduling, monitoring and fault tolerance purposes; an example of this type of technology is MapReduce [32].

This chapter shows that by appropriately integrating the above technologies, one can envision services for end users to conveniently launch their scientific applications on multiple clouds with minimal (if any) programming, deployment and management efforts; i.e., perform *sky computing*. In addition, it addresses the open question of whether the execution overheads added by these technologies would introduce performance or scalability penalties that mitigate their stated advantages. In this context, this chapter makes the following contributions:

- It shows that VM-enabled IaaS clouds, network virtualization and MapReduce paradigm [32] can be combined to manage a large-scale distributed execution environment using resources across distinct administrative domains connected by a wide-area network (WAN). Important bioinformatics applications based on BLAST [6] illustrates the potential of this approach, especially when resources from a single provider are not sufficient to solve the scientific problem at hand in a timely manner.

- It experimentally validates the approach on the Science clouds [54] by deploying a Xen-based virtual cluster across three sites, at the University of Chicago (UC), the University of Florida (UF) and Purdue University (PU). Nimbus virtual workspaces toolkit [53] has been chosen to authenticate users, to deploy VMs and to perform the initial configuration of the distributed environment. ViNe [133] has been used to connect the nodes behind NATs, over a 200Mbps WAN link.

- It evaluates the approach by comparing its performance with both purely sequential implementations and a leading MPI-based solution (also proposed and deployed for the first time by this work on a non-emulated WAN-based virtual network crossing distinct administrative domains without the use of handcrafted scripts for dealing with execution environment heterogeneity).

- It evaluates the overhead introduced by virtualization and MapReduce by comparing the performance of the BLAST application using physical and virtual machines, on LAN and WAN clusters, and with Hadoop implementation of MapReduce (CloudBLAST) and MPI (mpiBLAST).

- It proposes a skewed distribution of tasks using the Hadoop MapReduce framework to improve time-to-solution of a job in the presence of performance imbalance between the different cloud resources.

Experimental results show that the overheads of the above-mentioned technologies (virtual workspaces and ViNe) are insignificant, thus demonstrating the feasibility and benefits of executing BLAST on virtualized resources and across sites. CloudBLAST and mpiBLAST presented similar performance with a small advantage to CloudBLAST in a scenario where the target database fits in memory.

Distributed Bioinformatics Computing

Bioinformatics is being confronted with increasingly larger data sets of biological sequences produced by new massively parallel sequencing technologies (e.g., Roche 454, Applied Biosystems SOLiD, and Illumina HiSeq 2000), leading to computational jobs that take unacceptably long times if executed on a small number of machines. For these cases, distributed computing on multiple clusters at different locations is becoming an attractive, if not necessary, approach to achieve short execution times.

Grid-computing approaches have been developed with the primary goal of securely enabling the use of distributed pools of resources to allow the execution of large jobs. However, reconciling the policies and execution environments offered by distinct Grid and HPC resource providers as well as end user application requirements proved to be complex, time-consuming and expensive. The use of machine virtualization technology by IaaS cloud providers opened the possibility for end users to completely control the software stack in VMs as well as access to them, while still allowing resource providers to control resource allocation. This separation of responsibilities turned cloud computing highly popular.

Users of bioinformatics tools are typically unprepared to take advantage of these and other emerging technologies for distributed computing. One reason is the need to recast sequential tools as distributed jobs. Another reason is the challenge of understanding Grid and cloud technologies well enough to create and manage a distributed environment, which may require changes or additions of system software, and capabilities to schedule tasks, recover from failures and monitor distributed tasks.

More specifically, a bioinformatics researcher would face the following interrelated tasks:

- Find ways to decompose a large sequential application into smaller units so that it can run as multiple concurrent computational tasks.
- Identify, allocate and prepare machines with the necessary software environment for the execution of each of the parallel jobs and the storage of datasets.
- Ensure that machines are networked so that the necessary communication among concurrent jobs can take place.
- Deploy and manage the parallel execution of the application on distributed resources.

In the following subsections, the options available for each of the tasks are discussed, and the choices proposed by this chapter are identified on the basis of their suitability for integration

into a combined solution with good performance, low management complexity and minimal overhead.

Parallel Execution of Applications

Though many bioinformatics tools are parallelizable, parallelism is usually not explicitly expressed in their original code. Users interested in executing an application in parallel are required to either (a) develop or adapt existing software to distribute and manage parallel jobs, or (b) modify existing applications to make use of libraries that facilitate distributed programming such as MPI, OpenMP, RPC and RMI. When modifying applications, the effort is recurrent since new versions of the original sequential code may render the parallelized application obsolete. The amount of work involved may lead to parallel versions that lag and lack in features of the latest sequential tool version. Typically, modified applications will not include mechanisms to automatically handle failures.

Often, users opt for approach (a) because it is too expensive to modify existing code or source code may not be available. This is particularly true when it is relatively easy to run the application on multiple resources, each of which executes the sequential application on a subset of the inputs. Nevertheless, this solution requires additional software to manage and monitor job distribution and possibly offer fault tolerance: [9][16][76][95][125] are a small sample of Grid-based solutions for bioinformatics applications that automate the process of transferring or sharing large files, selecting appropriate application binaries for a variety of environments or existing services, managing the creation and submission of a large number of jobs to be executed in parallel, and recovering from possible failures. This work advocates the use of the MapReduce framework [32], which combines many features and lessons learnt from the Grid solutions.

To better depict the process of executing an application in parallel, it is useful to consider a specific widely used tool - BLAST [6]. BLAST finds regions of local similarity between

nucleotide or protein sequences. Given a set of k sequences $S = \{s_1, s_2, \ldots, s_k\}$, the program compares each sequence s_i to a database of n sequences $D = \{d_1, d_2, \ldots, d_n\}$ and calculates the statistical significance of matches. Two usual approaches to execute BLAST in a distributed environment (Figure 3-1) are: (a) to partition the input sequences so that each of separate instances of BLAST (called workers) looks for similarities of a subset of S, or (b) to partition not only the input sequences but also the database D. The second approach allows the database to grow and still fit on the local disk or local memory of worker nodes, at the cost of workers needing to communicate among themselves to generate the final combined comparison result.

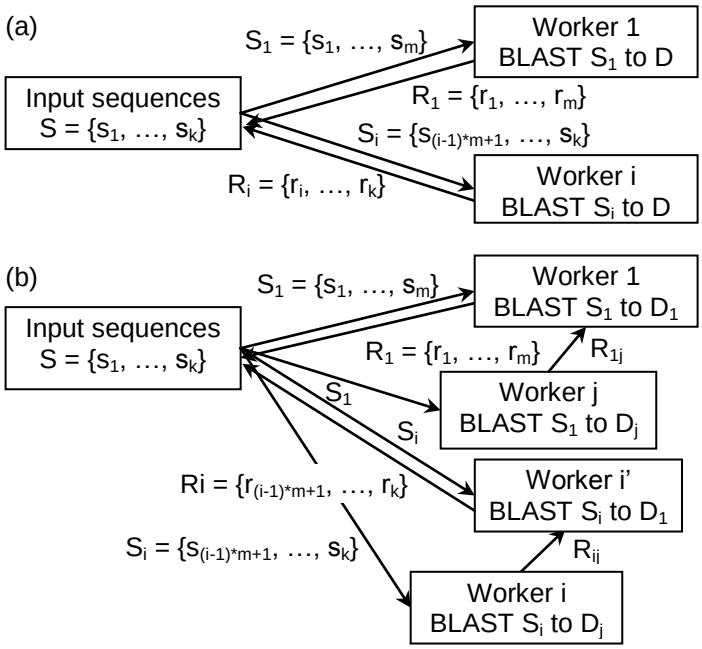


Figure 3-1. Alternatives for executing BLAST in parallel. In the first case (a), input sequences are partitioned into *i* subsets, each of which is processed by a worker, and results are concatenated in order. In the second alternative (b), input sequences are partitioned into *i* subsets and the database is partitioned into *j* subsets, each pair of subsets is processed by a worker, results from each database partition are sorted and merged, and results are concatenated in order.

The first approach (i.e., partitioning only the input sequences) can be easily implemented using a scripting language. The script should identify the input sequences for creation of input subsets, execute BLAST on distinct resources, and concatenate the result. For example, assuming

an underlying shared file system such as a Network File System (NFS), a bash script would simply use csplit, ssh/rsh, and cat commands (Figure 3-2). However, in practice, users still face the following challenges: (a) finding the ideal number of workers to use (or that are available), (b) load balancing and providing balanced partitions, since the time of BLAST execution is highly dependent on the length of an input sequence, (c) monitoring worker's health and the progress of the computation, and (d) recovering from the potential failure of some workers in order to avoid obtaining partial or corrupted results.

```
#!/bin/sh

#Input file name given as script argument
INFILE=$1
# Total number of input sequences
NOS=`grep "^>" $INFILE | wc -l`
#Total number of processors (hostsfile contains the host names in separate lines)
NOH=`cat hostsfile | wc -l`

#Split the input file into one-sequence files
csplit -z -s -n 1 $INFILE "/^>/" {*}
#Create an input file for each host recombining the one-sequence files
for ((i=0; i < $NOS; i++)); do
  c=$(( i % NOH )); cat xx$i >> subset_$c; rm xx$i
done
#Execute one input file per defined host
let c=0
pids=""
for h in `cat hostsfile`; do
  ssh $h "export BLASTDB=/db; blastx -d nt -i subset_$c -o out_$c" 2>&1 1>ssh_$c &
  pids="$pids $!"
  let c++
done
#Wait for jobs to finish execution
for job in $pids; do
  wait $job || { let "FAIL+=1"; echo "job $job failed. Total: $FAIL"; }
done
#Combine output
for ((c=0; c < $NOH; c++)); do
  cat out_$c >> final_out; rm out_$c
done
```

Figure 3-2. Bash shell script for executing BLAST in parallel. Given the BLAST input file name (as input argument) and a list of hosts to run the application (hostsfile), the script finds the number of input sequences and the number of workers, splits the input file into single-sequence files using csplit, recombines the sequences into subsets with approximately the same number of sequences, executes BLAST of each subset on remote resources, and combines the individual results after waiting for all workers to finish.

To handle these problems more efficiently, the use of MapReduce for this category of embarrassingly parallel applications is shown next. MapReduce as a software framework for distributed computation with fault tolerance was proposed by Google, inspired by functional languages, to realize parallel computing of data-intensive applications on a large number of unreliable resources. The programmer is required to implement a map function and a reduce function. The map function takes input records and generates intermediate key and value pairs. The reduce function takes all the values for an intermediary key and generates the result of processing these values. MapReduce framework is responsible for automatically splitting the input, distributing each chunk to workers (called mappers) on multiple machines, grouping and sorting all intermediary values associated with each intermediary key, passing these values to workers (called reducers) on multiple resources, and monitoring the execution of mappers and reducers as to speculatively execute or re-execute them when misbehavior or failures are detected. These characteristics greatly simplify deployment and management of jobs since jobs are submitted and controlled from a central location.

A classical example [32] of using the MapReduce framework is that of counting the number of occurrences of words in a large collection of web documents. The mapper analyzes a key/value input containing the document name as key and the document content as value, and for each word *w* in the document, a key/value pair *w/1* is generated. The reducer receives a collection of one (1) values for a particular word *w*, which can be added to produce the desired count for that word.

One possible setup to run the sequential BLAST program in the MapReduce framework is to assign to each mapper the execution of the BLAST operation for a subset of sequences, a map-only job. In other scenarios using BLAST or other bioinformatics applications, the reducer may

be useful – for example, the user may require the output of BLAST to be classified or ordered in some manner.

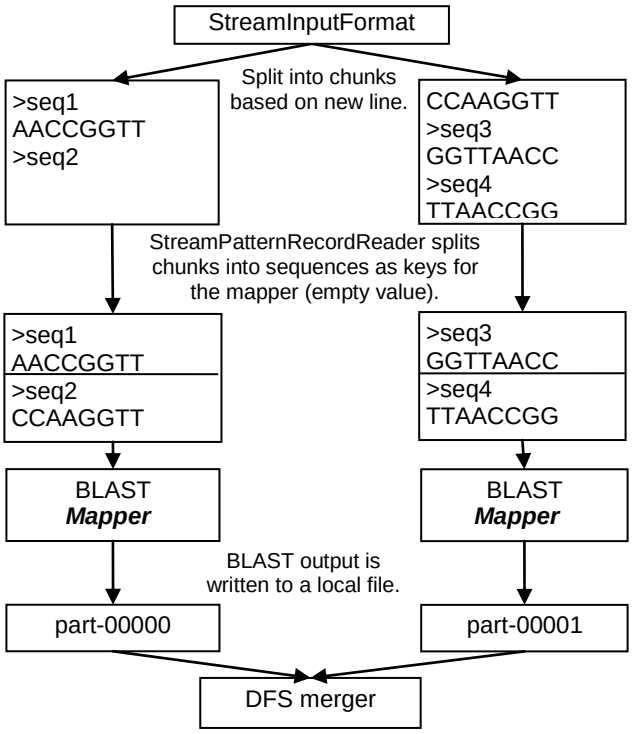


Figure 3-3. BLASTing with MapReduce. Given a set of input sequences, Hadoop splits the input file into chunks of same size, possibly splitting in the middle of a sequence. StreamPatternRecordReader was developed to interpret correctly the boundaries of sequences, which are passed as keys (of key/value pairs) to mappers that execute BLAST. DFS retrieves all local file outputs combining into a single result.

In this work, Apache Hadoop [11], an open-source implementation of the MapReduce paradigm, was used to execute NCBI BLAST2 in parallel. NCBI BLAST2 is a sequential implementation of BLAST, made publicly available by the National Center for Biotechnology Information (NCBI). The approach for running BLAST in parallel consists of segmenting the input sequences and running multiple instances of the unmodified NCBI BLAST2 on each segment, as illustrated in Figure 3-1a. Hadoop offers a streaming extension that allows easy execution of existing applications written in languages other than Java by spawning an external process and redirecting standard input, output and error streams. To deal with the fact that

Hadoop splits the input file into chunks of approximately the same size independently of the boundaries of a nucleotide or protein sequence, a pattern record reader was developed to retrieve entire sequences to be used as keys of key/value pairs given to mappers. The values of the key/value pairs are not used in this case and are kept empty. Similarly, the result of executing the BLAST application is also returned as keys of key/value pairs and stored on the Hadoop Distributed File System (DFS). To combine all results, the merge command of DFS was used. This process is depicted in Figure 3-3.

Although only experiments with BLAST are shown, the proposed solution also applies to other applications with similar execution profile, such as HMMER [39], megablast [147] and other derivatives of BLAST.

Application Deployment and Maintenance

With a parallelized application at hand, the next step is to find resources with the execution environment required by the application (e.g., MPI, and Hadoop); this can be an arduous task as the resources on the Grid are naturally heterogeneous with respect to operating system and libraries supporting applications. For example, an MPI-based approach requires the same MPI library (e.g., MPICH, and OpenMPI), which in turn may require machines to run one of several possible operating systems and additional software (e.g., to support a particular type of network protocol or technology). Similarly, a Hadoop-based approach requires machines with the Hadoop runtime and file system software, which can only execute if one of several acceptable version of Java is also installed. Both approaches, MPI and Hadoop, require secure shell (ssh) software for starting the distributed runtime environment. In the approach envisioned by this work, machine virtualization enables the use of VM images that already encapsulate these environments, effectively becoming appliances for execution of bioinformatics applications.

The long-term viability and desirability of the proposed approach as a sustainable and effective solution stems from the following four facts:

1. VM solutions for PC servers have matured in recent years [117] with many open source and commercial offerings (e.g., Citrix Xen [18], VMware [137], Microsoft Hyper-V [92], and KVM [78]). An increasing number of commercially available computational infrastructures using VMs can be observed, including 3tera [1], Amazon Elastic Computing Cloud (EC2) [7], Engine yard [41], GoGrid [114], and Joyent [70].

2. For resource providers, VM technologies offer easier management with better control of the shared resources and better isolation among users. From the perspective of clients, the ability to run the operating system of choice on the remote resources makes it possible to homogenize, at least in terms of execution environment, the heterogeneous set of available resources. It is no longer necessary to prepare different application binaries to run in different execution environments.

3. Applications can be encapsulated into VMs and, since VM images are used as templates to deploy multiple copies when needed, only the template images need to be maintained. When application updates are necessary, a new template image is created and multiple copies are automatically distributed by the IaaS cloud management services [73]. This relieves end users and/or administrators, potentially in multiple sites, from application installation, maintenance and deployment tasks. Application users are freed from the need to deal with application installation, compilation and tests. Templates prepared by specialists (possibly PaaS providers) can be used, and biologists can concentrate on providing input data and interpreting the results, as exemplified by the mpiBLAST appliance available for Amazon EC2, and the Hadoop cluster appliance made available by this work at the science clouds [54] marketplace for Nimbus-enabled clouds.

4. Commercial (e.g., VMware vSphere [138]) and open-source (e.g., Nimbus [53], Eucalyptus [99], and OpenNebula [102]) software packages for virtual infrastructure management are emerging to facilitate the control of a collection of VMs. For example, the Nimbus context broker service [74], used in this work, enables the formation of a virtual cluster by establishing a secure channel for exchanging information among appliances. Figure 3-4 shows an example of Nimbus virtual cluster configuration file. It instructs Nimbus to instantiate multiple VMs with different roles from a single user request using the specified VM image, to enforce information flow according to dependencies between VMs with different roles, to prepare all VMs with the necessary user and host credentials (to enable remote secure shell among all VMs), and to configure and initialize specific applications or runtime environments. Nimbus provides hooks in predefined locations of the VM image to place scripts that are executed by the context agent. In this particular example, a generic script that executes shell commands was implemented to avoid altering the VM image during tests. The Hadoop configuration files are generated by the head node from the information collected by the context broker and distributed to other VMs before initializing Hadoop runtime and running a sample parallel ‘cat’ command. Similar scripts can be written to configure MPI runtime environment, NFS servers and clients, or job schedulers.

```
<?xml version="1.0" encoding="UTF-8"?>
<cluster xmlns="http://www.globus.org/2008/06/workspace/metadata/logistics">
  <workspace>
    <name>headwork-node</name>
    <image>fc8-i386-nimbus-hadoop-cluster-001.gz</image>
    <quantity>1</quantity>
    <nic wantlogin="true">public</nic>
    <ctx>
      <provides><identity /><role>hadoop</role><role>hadoopmaster</role></provides>
      <requires>
          <identity />
          <role name="hadoopslave" hostname="true" pubkey="true" />
          <role name="hadoopmaster" hostname="true" pubkey="true" />
          <data name="exec"><![CDATA[
#!/bin/sh
MASTER=`cat /etc/thishostip`
HADOOPBASE="/opt/hadoop-0.19.0"
# To guarantee that this is only executed by the master
if [ -f /root/this_node_is_hadoop_master ] ; then
  # Configuring Hadoop
  sed s/HEADWORKER/$MASTER/g $HADOOPBASE/conf/hadoop-site-template.xml > /tmp/tmp && \
  mv -f /tmp/tmp $HADOOPBASE/conf/hadoop-site.xml
  cp /etc/thishostip $HADOOPBASE/conf/masters
  grep -v $MASTER$ /etc/hostsfile > $HADOOPBASE/conf/slaves
  cat /etc/thishostip $HADOOPBASE/conf/slaves > /tmp/tmp && \
  mv -f /tmp/tmp $HADOOPBASE/conf/slaves
  cp $HADOOPBASE/conf/slaves /etc/hostsfile
  # Copy Hadoop configuration to all other nodes
  for l in `cat /etc/hostsfile`; do
    scp $HADOOPBASE/conf/masters $l:$HADOOPBASE/conf/ || echo "masters to $l failed"
    scp $HADOOPBASE/conf/slaves $l:$HADOOPBASE/conf/ || echo "slaves to $l failed"
    scp $HADOOPBASE/conf/hadoop-site.xml $l:$HADOOPBASE/conf/ || echo "site to $l failed"
  done
  $HADOOPBASE/bin/start-all.sh
  # Run cat sample application with hadoop
  /opt/scripts/runhadoopcat.sh /etc/hostsfile 4 40 &
fi ]]> </data>
      </requires>
    </ctx>
  </workspace>
  <workspace>
    <name>work-nodes</name>
    <image>fc8-i386-nimbus-hadoop-cluster-001.gz</image>
    <quantity>3</quantity>
    <nic wantlogin="true">public</nic>
    <ctx>
      <provides>
          <identity />
          <role>hadoopslave</role>
      </provides>
      <requires>
          <identity />
          <role name="hadoop" hostname="true" pubkey="true" />
      </requires>
    </ctx>
  </workspace>
</cluster>
```

Figure 3-4. Nimbus virtual cluster configuration file. The virtual cluster in the example is composed of four virtual machines, one of which is considered the head node. The head node is responsible for creating and distributing Hadoop configuration, initiating Hadoop runtime environment, and starting the execution of a parallel ‘cat’ application.

Resource Performance Heterogeneity

When leasing resources from multiple IaaS clouds to perform sky computing, end users often receive a collection of VMs with performance profiles that differ from one another, depending on the hardware infrastructure and the types of service of each provider. This hardware heterogeneity manifests clearly when comparing resources in different providers [139]. Even within a single provider, differences may be present due to the following reasons:

- The servers used by cloud providers are not homogeneous - when servers are added to a farm, new models are used.
- Different Service Level Agreements (SLAs) may be available (e.g., Amazon EC2 offers instances that differ in the number of compute units and amount of memory)
- VMs may share a single host and the load of a VM can have a negative impact on other VMs.
- New processor capabilities such as the Intel Turbo Boost technology [64] can dynamically change core frequency according to power and temperature being used.

When computing resources are homogeneous with respect to performance, a task is typically broken into equally-sized sub-tasks and distributed to workers. However, when heterogeneous resources are present, the worst performing node dictates the time to solve a problem (the time perceived by the end users), since all sub-tasks must finish. A simple yet powerful solution, advocated for example by MapReduce, is to divide work into fine-grained sub-tasks so that nodes receive an amount of sub-tasks that is proportional to its performance. The drawback is that, in general, small sub-tasks increase the number of scheduling and management operations required by the system, potentially leading to a poor overall system performance. Furthermore, in some situations it is not possible to breakdown the work in as many sub-tasks as desired.

In these situations, some solutions to cope with heterogeneity have been proposed. For example, mpiBLAST [131] equally divides half of the input sequences among all workers and as

each worker finishes, an increasingly smaller portion of sequences is given to the worker (the number of remaining sequences divided by the number of workers) until a certain limit is reached (e.g., five sequences).

In this work, a skewed distribution of tasks is proposed in order to allow sub-tasks to better fit idle slots while avoiding a burst of very small jobs. The skewed distribution approach was developed when observing that in some situations the maximum number of sub-tasks *m* (defined by the number of query inputs in the case of BLAST) did not meet the MapReduce criteria that the number of sub-tasks *k* should be much greater than the number of workers *n* when solving a problem *P*. The extremes cases for the number of subtasks, *n* and *m*, are not recommended as the system becomes dependent on the slowest node in the former case, and the latter case may suffer from parallelization overheads.

The proposed method takes a hybrid approach, as follows:

- Let s be a fraction of the problem P, and k be the number of tasks between the minimum n (number of workers) and the maximum m (number of inputs) number of tasks.
- Divide $s \cdot P$ in exactly $k/2$ sub-tasks.
- Divide the remainder $(1 - s) \cdot P$ into another $k/2$ sub-tasks.

The idea behind the approach is to allow larger sub-tasks (half of all tasks) to be started on all nodes. As larger jobs finish, smaller jobs start to fill idle slots, especially in the faster resources, leading to better balance in resource utilization. Although similar to the mpiBLAST approach, two differences can be observed: (a) the skewed distribution allows the small sub-tasks to have similar sizes allowing resources with same performance to receive similar amount of work, and (b) instead of dividing work based on the number of sequences, sub-tasks are formed based on the amount of work, i.e., the size of sequences in the case of BLAST.

Biological Application Database

Bioinformatics applications make use of biological data sets of ever increasing sizes, as shown by statistics from the protein sequence UniProtKB/TrEMBL database [42] (Figure 3-5). Periodical updates of these databases are necessary to offer up-to-date results to biologists. Propagation of these updates is not an easy task when dealing with thousands of resources.

Data management techniques in Grid computing (e.g., GVFS [148], and SRB [19]) could potentially be used to automate the distribution of updated data to VMs on the cloud. However, in general, bioinformatics applications are not designed and implemented to take advantage of Grid data management services. In addition, best performance is obtained when datasets are as close as possible to compute nodes, ideally in the local storage. For example, Hadoop DFS takes advantage of the local storage by replicating and distributing input and output data.

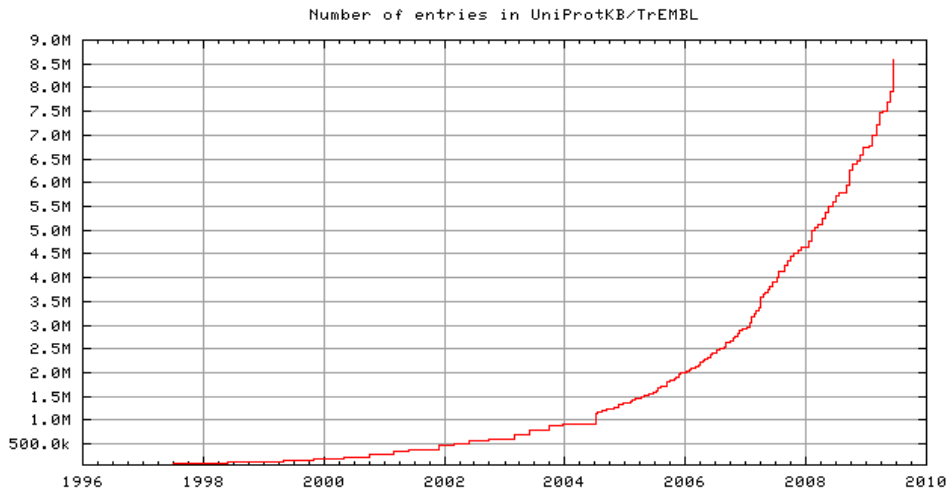


Figure 3-5. Growth of protein sequences in UniProtKB/TrEMBL database. Release 40.6 (16-Jun-2009): "it contains 8594382 sequence entries comprising 2774832018 amino acids. 1128426 sequences have been added since release 40, the sequence data of 2330 existing entries has been updated and the annotations of 4332439 entries have been revised" [42].

On a VM-powered distributed environment (as in this work), the proposed approach is to encapsulate biological data into VM images or into independent virtual disk images. Replications

of disk images are offered as services in cloud infrastructures, facilitating data maintenance as only a reduced number of images require manual update interventions and data replication is inherited from disk replication.

Parallel Applications Require Network Connectivity

Independently of the parallelization solution chosen, distributed applications require all computing nodes to be reachable with static IPs [130]. Having all-to-all communication among nodes is particularly challenging in multi-site multi-domain deployments due to the presence of NATs and firewalls.

To support distributed applications running in a large number of nodes spread across the Internet, different overlay networks architectures have been proposed, including NAT-aware network libraries and application programming interfaces (APIs) [33][82][83][121], virtual networks (VNs) [69][129][133], and peer-to-peer (P2P) systems [28][49][56].

ViNe approach [133] was chosen to provide the necessary connectivity among VMs deployed in different sites due to the fact that it does not interfere with intra-site communication – full LAN performance is available to the nodes on the same LAN. In addition, ViNe imposes low overhead on cross-site communication. ViNe routers are typically deployed at each site, and responsible for intercepting packets that cross LAN boundaries. Intercepted packets are transported to the destination (in private networks) using firewall and NAT traversal techniques.

One important advantage of ViNe over other solutions is its performance. For example, for the experimental setup described in the next section, ViNe routes packets at rates over 800Mbps, which is much higher than the available bandwidth (200Mbps). In terms of latency, less than 0.5 ms round-trip overhead on inter-site communication was observed (over the 28 ms physical network latency). As a result, the network virtualization overhead is mostly due to packet encapsulation (less than 5%).

Experimental Setup

To validate and evaluate the performance of the proposed approach, the Science Clouds [54] testbed was used. The testbed comprises multiple cloud providers configured in the academic space and providing different Service Level Agreements (SLAs) to the user; access is granted to scientific projects on a voluntary basis. Apart from providing a platform on which scientific applications explore cloud computing, the Science Cloud testbed creates a laboratory in which different IaaS providers use compatible technologies to provide offerings allowing experimentation with sky computing.

The sky computing study uses resources on three sites: University of Chicago (UC) [54], University of Florida (UF) [134] and Purdue University (PU) with SLAs and services as described in Table 3-1. All sites use the same virtualization implementation (Xen) and although the versions and kernels slightly differ, VM images are portable across sites. All sites use Nimbus [53], an EC2-compatible open source IaaS implementation. Because of that, the sites are also IaaS API-compliant, but, as Table 3-1 shows, they offer different SLAs. While all sites offer an "immediate lease", the provided "instances" (defined in terms of CPU, memory, and type of connectivity) are different. More significantly from the usability point of view, the UC and PU clouds provide public IP leases to the deployed VMs, while UF does not provide such leases, limiting global addressability. As previously described, ViNe was used to provide the necessary connectivity.

As a representative bioinformatics application, BLAST was chosen, as both sequential [6] and MPI-based parallel [31] versions are available. The MPI version uses the approach that segments not only the input sequences, but also the database to be more efficient when the entire database does not fit into memory. The CloudBLAST approach, which segments only the input sequences, has been developed using Apache Hadoop [11] as previously detailed.

Table 3-1. Service Level Agreement and "Instances" at Each Cloud Provider.

	UC	UF	PU
Xen VMM	3.1.0	3.1.0	3.0.3
Guest kernel	2.6.18-x86_64	2.6.18-i686	2.6.16-i686
Nimbus version	2.2	2.1	2.1
CPU architecture	AMD Opteron 248	Intel Xeon Prestonia	Intel Xeon Irwindale
CPU clock	2.2 GHz	2.4 GHz	2.8 GHz
CPU cache	1 MB	512 kB	2 MB
VCPUs	2	2	2
Memory	3.5 GB	2.5 GB	1.5 GB
Networking	Public	Private	Public

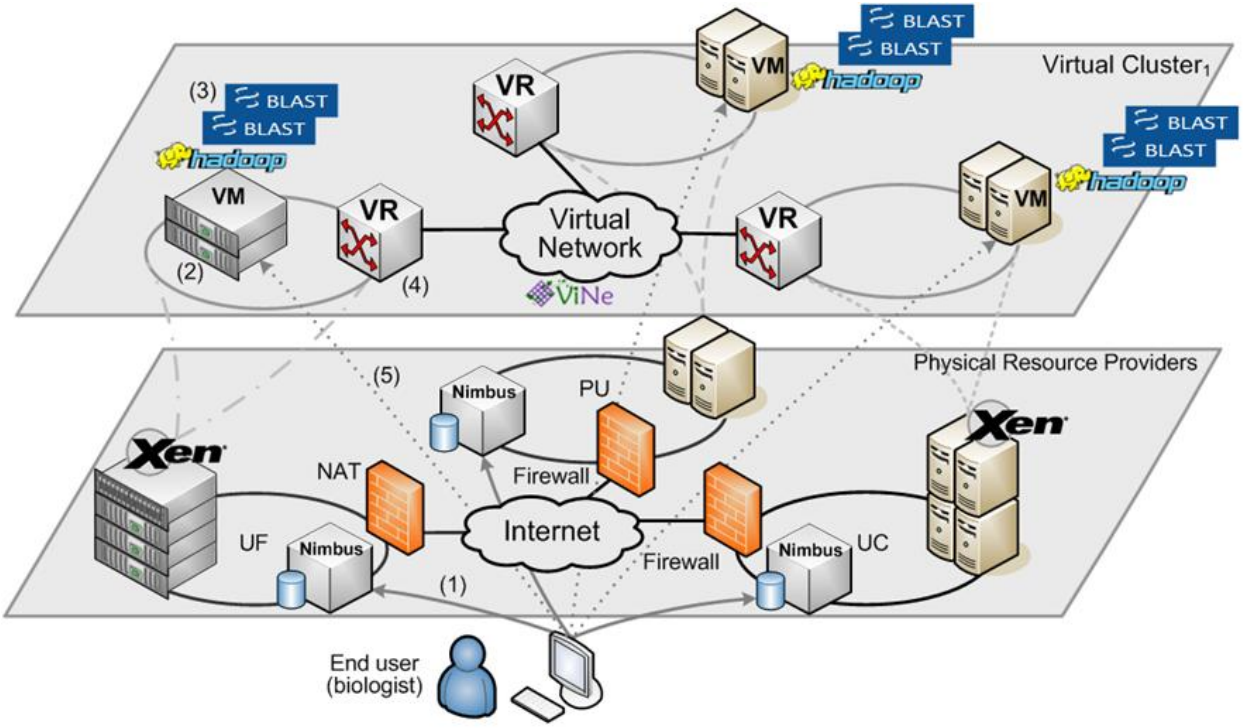


Figure 3-6. Virtual cluster creation following CloudBLAST approach. End user selects a VM image and requests instantiation of a virtual cluster using UF, UC, and PU IaaS providers (1). Nimbus selects available physical machines and starts cloning (2). Xen-based VMs are configured (3) and connected through ViNe VM clones (4). The biologist can execute Hadoop-based or MPI-based BLAST on the distributed environment (5).

All the necessary binaries (i.e., BLAST 2.2.18, mpiBLAST 1.5.0beta1, MPICH2 1.0.7, Hadoop 0.16.2, Java 1.6.0, and Python 2.5.1) were installed in a Xen guest domain image and uploaded to virtual workspaces image repositories at all sites. The non-redundant (NR) protein

sequence database from NCBI split into 1 fragment and 30 fragments (each with a total of 3.5 GB of data) was also placed in the VM image.

The process of deploying a virtual BLAST cluster consists of the following steps (Figure 3-6):

1. A VM image with the necessary software is prepared and uploaded to the virtual workspaces factory server. A biologist requests the creation of instances of the VM.
2. The virtual workspaces factory server in each site selects available physical nodes and starts the VM cloning process.
3. VM clones are booted and configured with appropriate network addresses. Network routes are adjusted to use Virtual Routers (VRs) to reach nodes crossing site boundaries.
4. A VR VM is cloned and started in each site, offering all-to-all connectivity among all VMs.
5. The biologist logs into one of the VMs, uploads the input sequence and starts executing BLAST jobs.

In the experiments conducted with BLAST, two sets of nucleotide sequences, each with 960 sequences were used as input. The blastx program from the NCBI BLAST suite, which compares nucleotides sequences against protein sequences by translating the nucleotides into amino acids, was used. Characteristics of the 2 input sets are shown in Table 3-2.

Table 3-2. Experimental Sequences Characteristics.

	960 long	960 short
# of sequences	960	960
Longest sequence	5,813	850
Shortest sequence	215	101
Standard deviation	515	170
Total number of nucleotides	1,072,143	425,735
Average nucleotides per sequence	1116.82	443.47

Evaluation and Experimental Results

Evaluation of the proposed solution is performed by answering six questions.

Question 1: Are there machine virtualization overheads?

To evaluate the overhead of machine virtualization, experiments with BLAST were conducted on physical machines (pure Linux systems without Xen VMM) and later on the Xen

VMs hosted by the same hardware. Only resources at UF were used for this experiment. As it can be observed in Figure 3-7, virtualization overhead is barely noticeable when executing Hadoop-based and MPI-based BLAST, varying number of processors allocated for the job from 8 to 64. In all experiments in this chapter, graphs show absolute speedup ($Speedup_p = T_{sequential\ \ UF}/T_{p\ parallel}$) of parallel jobs ($T_{p\ parallel}$ is the time-o-solution of a parallel job) relative to the execution time of a sequential run on a UF resource ($T_{sequential\ \ UF}$ is shown in Table 3-3). The observed good performance is due to the minimal need for I/O operations as a result of replicating the database in each node and fully loading it to main memory. It also confirms other observations of virtualization having negligible impact on compute-dominated applications [47] or even improving performance due to artifacts of implementation, e.g., file system double caching[1].

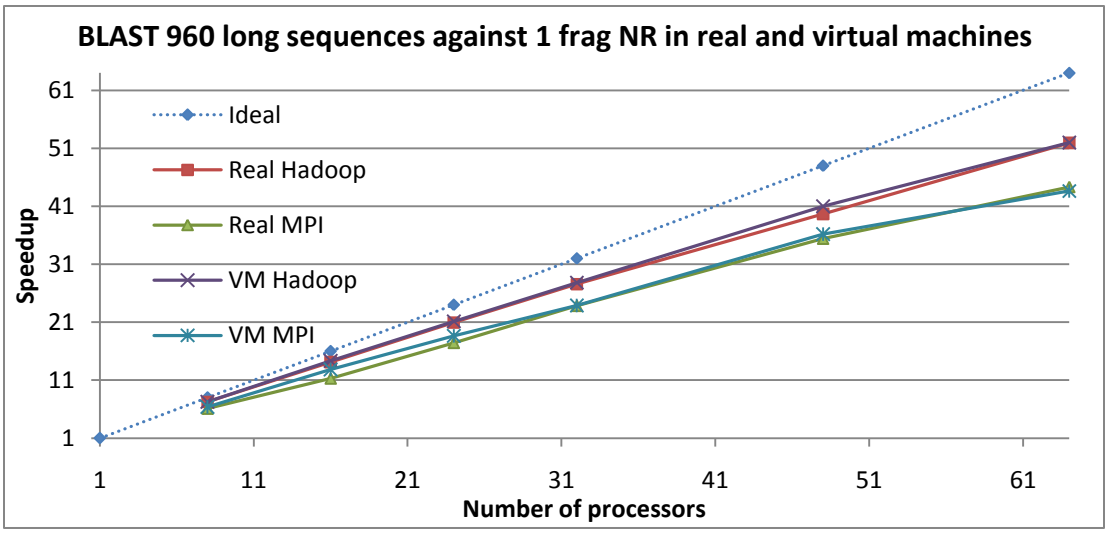


Figure 3-7. Comparison of BLAST speedups on physical and virtual machines. Speedup is calculated relative to sequential BLAST on virtual resource. The ideal line represents the linear speedup in an ideal scenario where no overhead is added by the parallelization procedure. VMs deliver performance comparable with physical machines when executing BLAST on a variety of configurations for both Hadoop and MPI versions of BLAST.

[1] Personal communication with Dr. Kate Keahey on 03/2006.

Question 2: Is the solution scalable with the increase in number of processors and considering time-to-solution metric?

To evaluate performance scalability of mpiBLAST and CloudBLAST, experiments with different numbers of nodes, processors, database fragments, and input data sets were executed. For this experiment, UC and UF clouds were used with exactly 2 processors in each node. Whenever possible, the numbers of nodes in each site are kept balanced (i.e., for experiments with 8 processors, 2 nodes in each site are used; for experiments with 16 processors, 4 nodes are used in each site, and so on). For experiments with 64 processors, 12 nodes at UC and 20 at UF were used due to limited number of available nodes at UC.

Results are summarized in Figure 3-8. For both long and short sequences input data sets, CloudBLAST shows a small advantage over mpiBLAST. For the input data set of long sequences, a 57-fold speedup can be observed with CloudBLAST when using 64 processors on 2 sites, while mpiBLAST delivers 52-fold speedup. The sequential execution time for the long input set of 43.76 hours on a single UF machine is reduced to 46 minutes on the cloud with CloudBLAST and to 50 minutes with mpiBLAST. Similarly, for the short input set, 17.1 hours of sequential execution is reduced to 19.5 minutes with CloudBLAST and 25 minutes with mpiBLAST. The performance difference between CloudBLAST and mpiBLAST is more noticeable when sequences are short, indicating an efficient implementation of input sequence split and job coordination by Hadoop. Two-site results show better speedup than one-site due to processors that are fast at UC (approximately 1.18 times faster). The publicly available mpiBLAST can be further hand-tuned for higher performance, but such optimized implementations are not publicly accessible[2].

[2] Personal communication with Dr. Wu Feng on 07/2008.

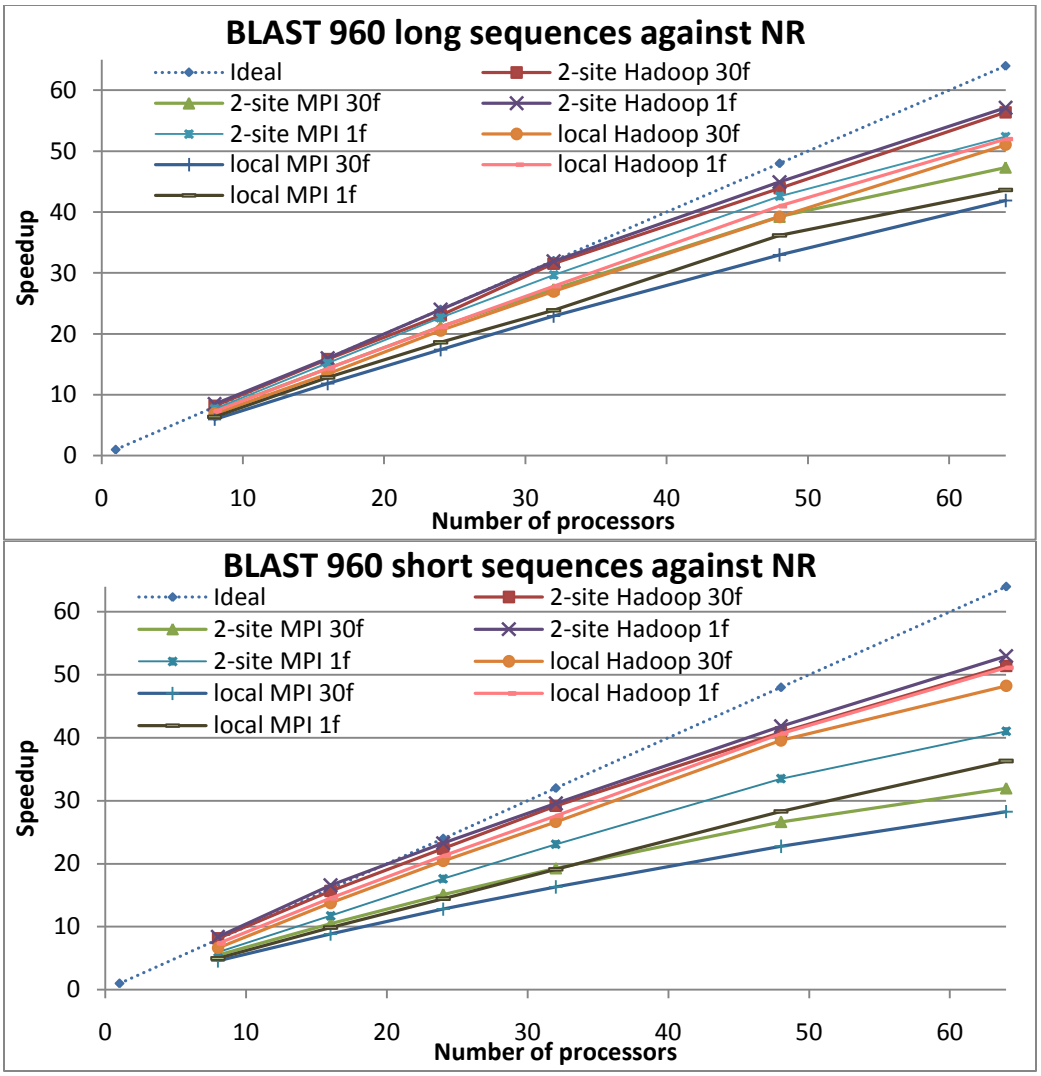


Figure 3-8. Speedup curves for CloudBLAST (Hadoop) and mpiBLAST (MPI). 960 short and long sequences are BLASTed against NR database segmented into 1 and 30 fragments on 1-site (UF) and 2-site (UC and UF) resources. CloudBLAST presents a slightly better performance than mpiBLAST, in particular when sequences are shorter. 2-site configuration shows higher speedup than the corresponding 1-site configuration due to superior performance of UC resources. The increase in the number of BLAST database fragments raises the number of I/O operations, resulting in lower performance.

Question 3: Are there impacts due to network virtualization?

Although the entire target database fits into memory in the setup used, runs with the database segmented into 30 fragments were performed in order to increase the amount of network communication for mpiBLAST runs (Figure 3-9) and potentially stress ViNe overlay.

Speedup graphs (Figure 3-8) show that for both CloudBLAST and mpiBLAST, the performance of BLAST decreases using more fragments due to the increase of map and unmap operations performed. The same graph shows that the difference between executions with different fragmentation is the same for local and 2-site scenarios, indicating no impact on the performance due to VN.

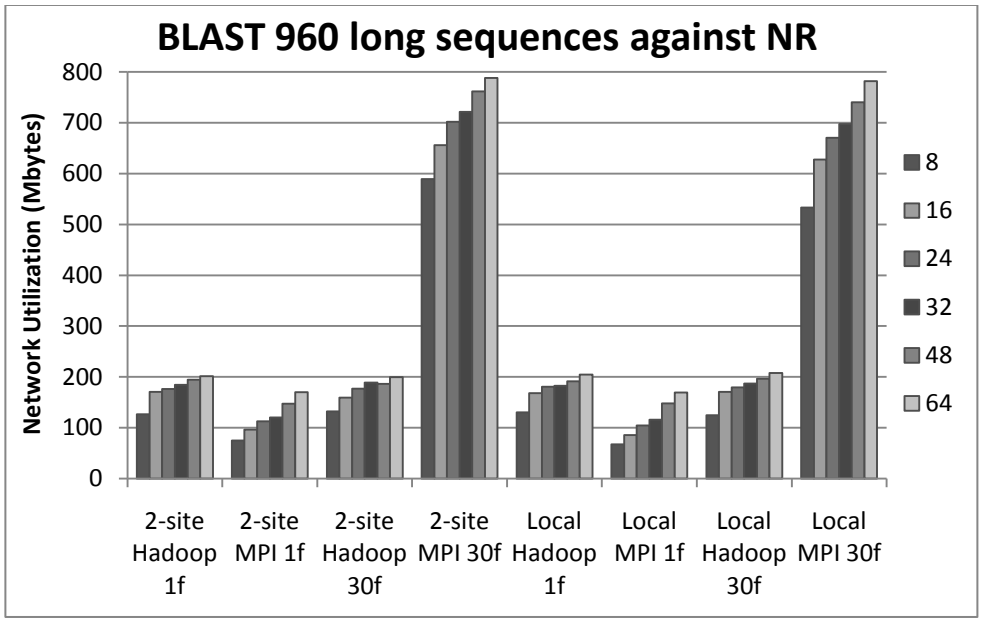


Figure 3-9. Network usage for CloudBLAST (Hadoop) and mpiBLAST (MPI). Experiments BLASTing 960 long sequences against NR database segmented into 1 and 30 fragments on 1-site and 2-site resources (varying from 8 to 64 processors) show low network usage by BLAST when compared to CPU utilization. Increase in network usage by mpiBLAST, when increasing the number of database fragments, was expected due to the need to combine output from various workers.

Question 4: How MPI and Hadoop BLAST perform when normalizing resources according to sequential execution performance?

The SLAs expressed as "instances" from each provider of the meta-cloud (as described in Table 3-1) are different not only in terms of CPU speed, but also CPU architecture, memory and cache, which makes it difficult to compare performance of instances from different providers. To establish a comparison base, the performance of the sequential execution of BLAST with an

input of 960 long sequences was selected as a benchmark. Resources in three sites were used for this experiment, and according to the sequential job execution, a normalization factor was established (Table 3-3). For example, an experiment with 10 UF processors, 10 UC processors and 10 PU processors would be expected to provide the performance of a cluster with 34.24 UF processors. These normalization factors are used to normalize the number of processors in Figure 3-10, which compares the speedup in 1-site (UF) and 3-site (UC, UF, and PU) configurations. According to the availability of resources, the distribution of the number of processors involved in the 3-site experiments varied according to Table 3-4.

Table 3-3. CPU normalization factor established using BLAST sequential execution time.

	UC	UF	PU
Sequential Execution Time (hours:minutes)	36:23	43:06	34:49
Normalization Factor	1.184	1	1.24

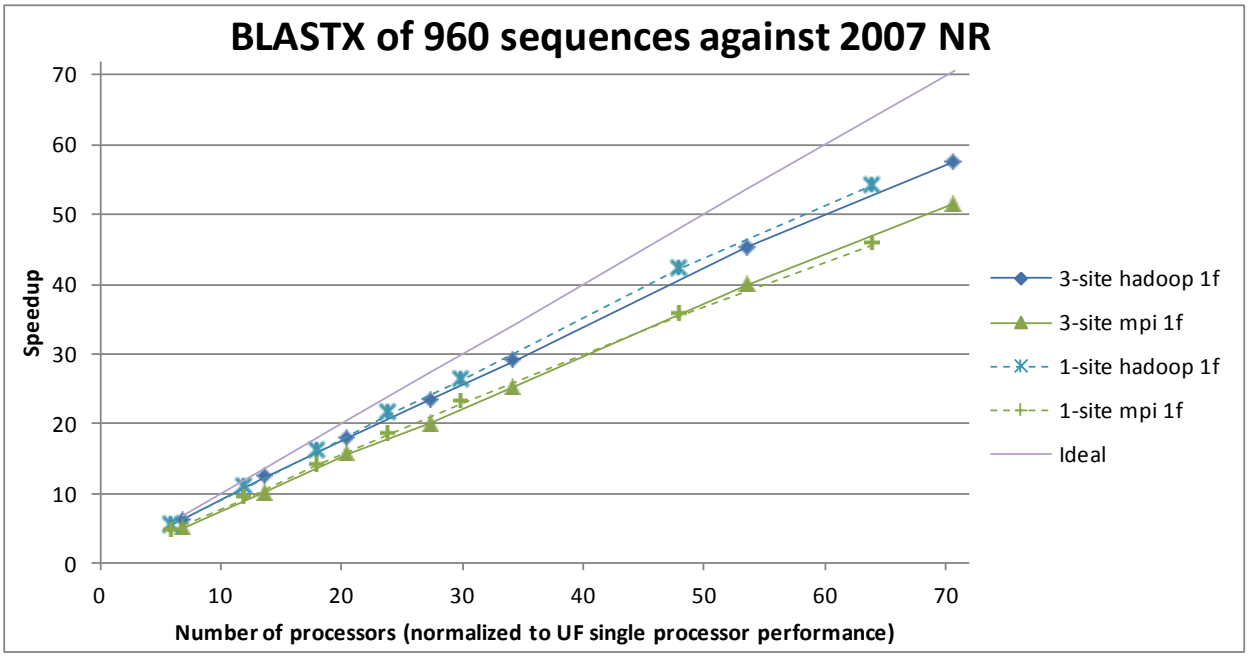


Figure 3-10. Speedup curves on different number of sites with normalized number of processors. Both single site (UF) and 3-site (UC, UF and PU) configurations shows good speedups when running Hadoop and MPI versions of BLAST in a virtual cluster, with 3-site performance comparable to that obtained using a single site.

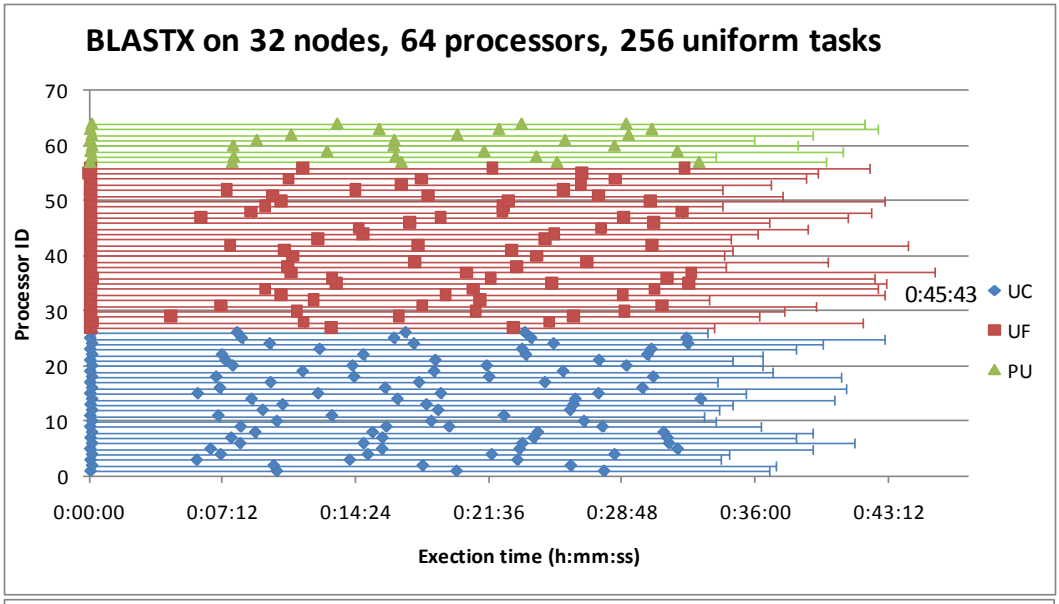
Table 3-4. Virtual cluster distribution for the 3-site experiment configuration.

Experiment	UC	UF	PU	CPUs	Normalized CPUs
1	2	2	2	6	6.848
2	4	4	4	12	13.696
3	6	6	6	18	20.544
4	8	8	8	24	27.392
5	10	10	10	30	34.24
6	20	20	8	48	53.6
7	26	30	8	64	70.704

Question 5: What are the differences between uniform and skewed task distribution?

Looking more closely at the configuration with 64 CPUs distributed across three clouds, Figure 3-11 shows the execution time of each of the 256 sub-tasks of a 960-sequence BLAST job on 64 CPUs with (a) uniform and (b) skewed distribution of tasks. The progress of execution time is shown in the horizontal axis, while each of the 64 CPUs is represented in the vertical axis with different colors indicating the location of the CPU. The marks on the graph indicate the moment each task initiates execution, and the vertical bars indicate the moment the last task in each resource finished execution. When sub-tasks have similar sizes (uniform distribution), workers receive between 3 and 5 sub-tasks and the overall execution time (45.72 minutes) is dictated by the slower resource (UF). With the skewed task distribution, the time-to-solution is reduced to 42.1 minutes, with UF resources getting on average 3 sub-tasks, while UC receives 5.2 sub-tasks and PU receives 4.25 sub-tasks. Shorter tasks are executed in the end allowing UC and PU resources to accommodate more tasks when compared to UF. Although the gain in this case is minimal, it is expected that the incremental efficiency of skewed distribution will become more evident in larger scale experiments or when resources demonstrate greater performance heterogeneity – e.g., the EC2 minimum compute unit is approximately half of a single UF VM instance.

(a)



(b)

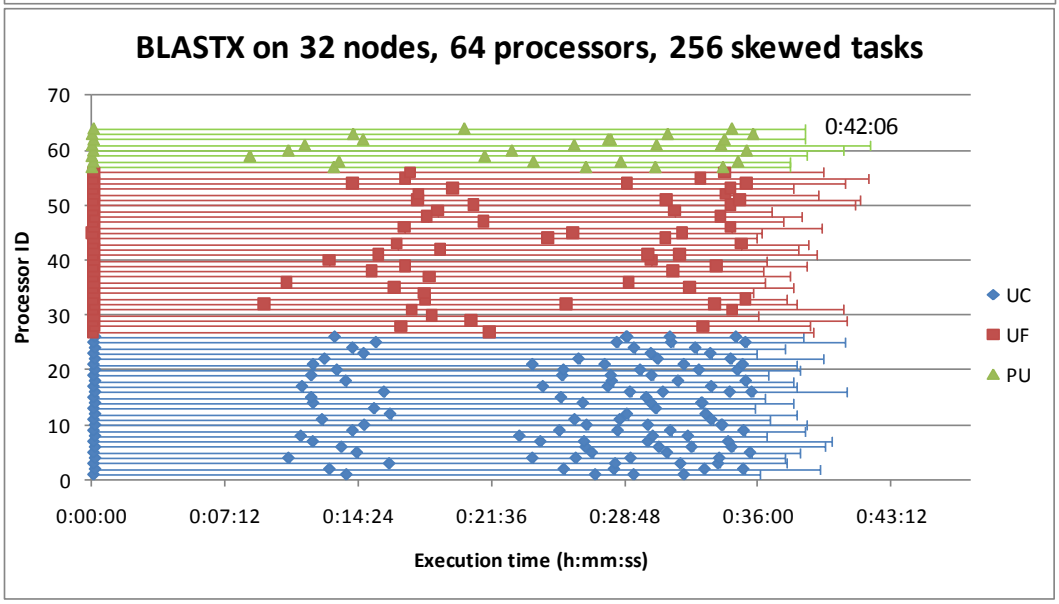


Figure 3-11. Comparison between uniform and skewed distribution of Hadoop BLAST tasks. Graphs show two executions of a 960-sequence BLAST job on 64 processors across 3 different sites (UC, UF and PU). 256 tasks in both runs are divided with (a) uniform and (b) skewed distribution. The progress of time is shown in the horizontal axis while the vertical axis represents each of the 64 individual workers (different colors depict the type and location of worker's resource). In this particular experiment, the overall time perceived by the end user when running with skewed tasks is 8% shorter than when running with uniform tasks.

Question 6: How easy is to accommodate a new version of the sequential BLAST with Hadoop BLAST?

To demonstrate that Hadoop BLAST can adapt to new versions of the sequential application, experiments with a new version of BLAST (2.2.19+) were executed as soon as it became available. To integrate the new version, the BLAST VM image was updated with the new binaries and the user was simply required to change executable names and arguments specification when submitting a job. No modifications to Hadoop or to the additional extensions were required. Only resources at UF were used in this experiment to compare execution of 960 long sequences with BLAST 2.2.18 and 2.2.19+. Figure 3-12 shows that CloudBLAST was able to benefit from the improvements made to the BLAST algorithm while continuing to provide good scalability and without requiring redevelopment of the parallel execution environment, as it would be the case for mpiBLAST.

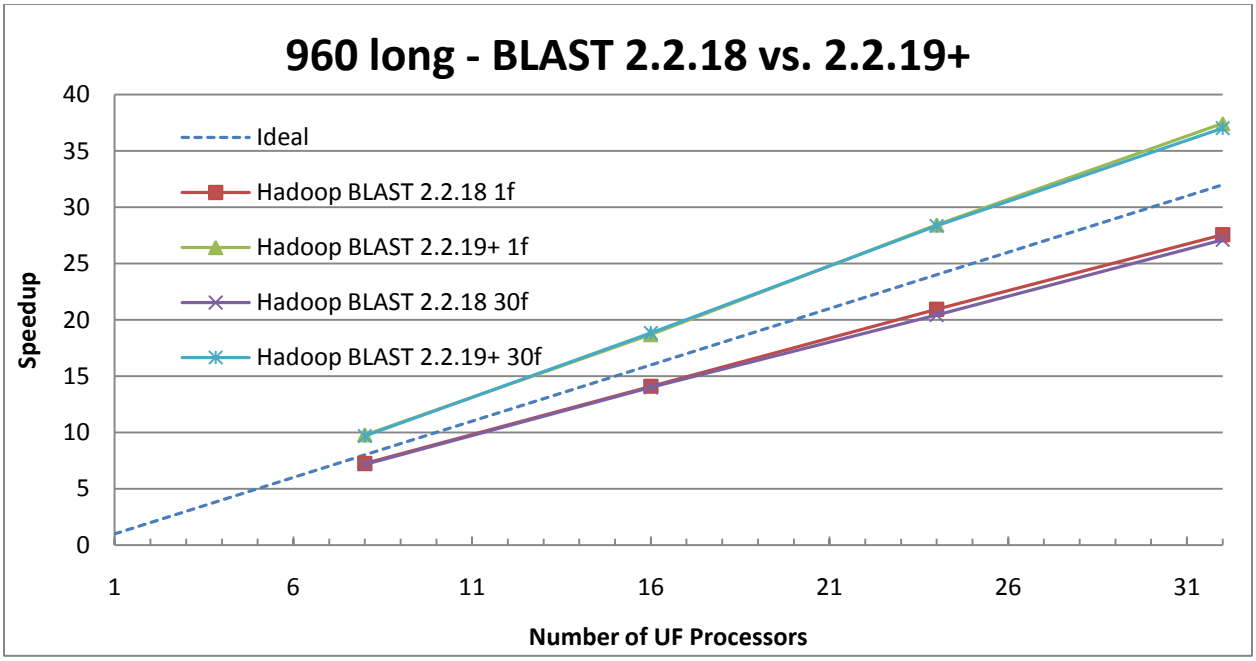


Figure 3-12. Speedup curves comparing performance of two different versions of BLAST. It shows that CloudBLAST can readily take advantage of improvements to the sequential BLAST algorithm. In this particular case, the improvements in the newer version (2.2.19+), when compared to version 2.2.18, using 960 long sequences as input on UF resources.

Summary

The research reported in this chapter investigated an efficient approach to the execution of bioinformatics applications and validated it by demonstrating its low overheads and high performance by developing CloudBLAST, a distributed implementation of NCBI BLAST.

The crux of the approach is the integration of the MapReduce approach for executing unmodified applications in parallel with the encapsulation of software environments and data in virtual machines connected by virtual networks that offers all-to-all connectivity.

The experiments were all conducted in a real deployment, with Internet-connected IaaS clouds deployed at the University of Chicago, the University of Florida and Purdue University. Scaling-out across administrative domains was made possible by the use of network virtualization. Management scalability, instantiating VMs in separate clouds and automatically configuring them as a virtual cluster, was made possible by the use of simple scripts and the Nimbus contextualization service. Using the same infrastructure, both CloudBLAST and mpiBLAST offered good performance scalability. The exception was in the presence of VM failures. CloudBLAST was able to output the entire result with a small increase in the overall execution time, while mpiBLAST was not able to complete the job. CloudBLAST has also advantages in terms of development, management and sustainability, since it easily accommodates releases of new versions of the sequential NCBI BLAST. Surprisingly, both mpiBLAST and CloudBLAST showed performance gains with increases in the number of available processors from geographically-distant locations while making use of virtualization technologies. No noticeable difference in execution time was observed when allocating all resources from one site versus allocating half of the resources from each site, even in small scales (e.g., 4 processors). An additional increment in CloudBLAST performance was obtained with skewed distribution in the presence of resource performance heterogeneity and high number

of workers. All software components used in this work are open-source and readily available from the respective project sites, and a Hadoop cluster VM image with the software stack proposed has been made available through Science Cloud marketplace.

At a more general level, the three main conclusions from the work reported in this chapter are as follows:

- For applications whose dependency structure fits the MapReduce paradigm, the CloudBLAST case study suggests that few (if any) performance gains would result from using a different approach that requires reprogramming. Conversely, a MapReduce implementation, such as Hadoop, brings with it significant advantages from the perspective of management of failures, data and jobs, with an incremental improvement from the skewed distribution proposed in this work. This conclusion reinforces similar claims of proponents of the MapReduce approach and demonstrates them in the context of bioinformatics applications. Although only validated with BLAST, the proposed approach also applies to other embarrassingly parallel applications with text-based inputs, such as HMMER [39], megablast [147] and other derivatives of BLAST. The Hadoop record reader developed to locate the boundaries of biological sequences can be reused by other applications with input limited by a textual pattern, but development of new record readers may be necessary for applications with complex input formats.

- Using virtual machines with software and data needed for execution of both the application and MapReduce greatly facilitates the distributed deployment of sequential code. The middleware used for creation, cloning and management of virtual machine images can be presented to users as a service that transparently provides environments with enough resources for the size of the problem to be solved. In particular, facilities such as the Nimbus contextualization service proved to be vital for automating the configuration and initialization of the execution environment across multiple resources.

- The use of virtual networks is essential when VM providers belong to different administrative domains behind NATs/firewalls. They must also have good performance in order to take advantage of physical speeds of both local and global network connections. The use of the VN could be transparent to the user when providers use it to connect machines in different facilities. Alternatively, users could use the VN to connect resources leased from different providers. In both cases, VNs greatly facilitate the scaling out of applications which, like those found in bioinformatics, use increasingly larger data sets. Furthermore, the all-to-all connectivity offered by VNs make it possible to run applications developed for local clusters (e.g., mpiBLAST and other MPI-based applications) without the need to develop site-specific scripts that deal with the typical single point of entry configuration offered by providers.

This chapter assumed that end users have previous knowledge of the application resource consumption when requesting the allocation of resources for running a large computation on

multiple clouds, and demonstrated the feasibility of proposed approach. The next chapter looks into how to systematically predict resource requirements of applications in order to improve the scheduling of applications.

CHAPTER 4
APPLICATION RESOURCE CONSUMPTION PREDICTION

Resource consumption by an application in the form of CPU time, amount of memory, network bandwidth, and disk space consumed, is a useful piece of information when available before execution. It can be used by schedulers to accommodate the maximum number of applications without resource contention, it can help estimate the waiting time on queued systems, it can identify the best resource to run an application and analyze what-if scenarios, or it can provide an estimate of the cost of running an application on pay-per-use facilities (e.g., a cloud). Nonetheless, this information is often not available to users or computational systems. In cases where predicted resource consumption is provided, the prediction rarely takes into consideration both application characteristics and system performance. The use of Machine-Learning (ML) algorithms to predict application resource consumption is an appealing approach that has been pursued by previous studies [3][37][50][62][72][93][94][96][108][118][119]. These studies have proposed the use of specific ML algorithms applied to scenarios ranging from generic jobs in batch systems to specific applications that could be part of a workflow distributed across a grid of resources. The abundance of proposed solutions, the diversity of dataset attributes and prediction performance metrics, combined with limited availability of comparative evaluations, make it difficult for users and system developers to choose an appropriate prediction method. To remedy this situation, this work makes the following contributions:

- It identifies the best ML algorithm and provides reasoning for the results, for predicting execution time, memory and disk requirements for two bioinformatics applications, namely BLAST [6] and RAxML [124], on different types of computer clusters. The regression algorithms considered in this study include k-nearest neighbor (k-nn) [72][119], linear regression [3], decision table [108], Radial Basis Function network (RBFn) [37], Predicting Query Runtime (PQR) [62], and Support Vector Machine (SVM) [22][36]. The datasets comprising more than 2000 CPU-hours of execution are made publicly available for future research.

- It proposes the Predicting Query Runtime Regression (PQR2) algorithm, a generalization of the PQR classification tree approach, to the regression problem. The extension consists of adding regression functions at the leaves of the PQR tree in order to provide fine-grained prediction.
- It investigates the impact of attributes on prediction accuracy and makes the case for increasing the number of attributes (data space), in particular taking into account heterogeneous systems performance and detailed application-specific characteristics.

Motivation

In order to fairly share computational capacity among users, a provider of processing cycles, such as an IaaS cloud provider or an HPC center, typically makes use of scheduling algorithms for improving the utilization of the entire system while still offering expected levels of service to its users. For providers who run mainly long-running computationally intensive applications, space-sharing of resources is preferred to the time-sharing alternative to avoid system performance degradation. Degradation originates from excessive context-switch and page faults. Space-shared systems can be modeled as a resource manager and a scheduling server controlling how the computing resources are to be used by incoming job requests that are maintained in a queue (Figure 4-1). The resource manager is responsible for collecting information about the available resources in the system, monitoring the current state of resources, and providing this information to the scheduler. The scheduler in turn makes decisions about which job is to be executed in a certain resource. The datacenter resources could consist of clusters from several generations with different architectures, processing speeds, amount of memory, interconnection, storage, operating systems and libraries. When submitting jobs, users are often asked to provide information about the job resource requirements. Requirements can include number of hosts, number of processors per machine (PPM), amount of memory per processor (MPP), and amount of time to execute the job. This information serves as guidance for the scheduler to obtain maximum utilization of the datacenter resources without creating

conflicts among jobs. At the same time, users can select a resource as specific as necessary for their jobs.

Job	Host	PPM	MPP	Time
J1	2	1	500MiB	4
J2	5	1	1GiB	1
J3	1	1	2GiB	2
J4	3	2	800MiB	3
J5	1	1	300MiB	6
J6	3	1	650MiB	3

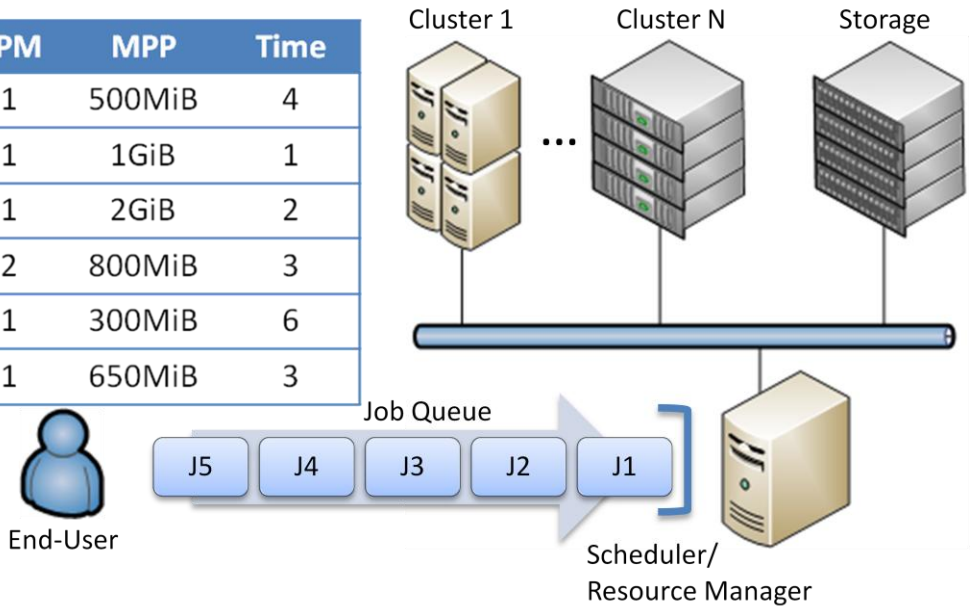


Figure 4-1. Typical system architecture with queue-based resource management. End-users submit requests to resource managers that are also responsible for tracking state of resources and submitting jobs for execution. The requests with a number of resource requirements are maintained in a queue. The scheduler is responsible for ordering the requests according to established policies.

Currently, most IaaS cloud providers offer a provisioning model based on immediate or best-effort lease. However, given the advantages just described, management toolkits that integrate schedulers and allow for the formation of a queue are starting to become available [122]. Various scheduler heuristics have been proposed over the past decades for online scheduling of jobs in space-sliced high-performance clusters, each with its own benefits and drawbacks. Scheduler algorithms can be classified mainly according to three characteristics: queue-priority, backfilling, and preemption. The queue-priority policy defines the order of the jobs in the queue. The backfilling policy defines the jobs that can be scheduled ahead of time when there are not enough resources available for the first job in the queue to be executed, but there are still unused resources for other jobs in the queue. The preemption policy defines which

running jobs should be seized to allow the execution of a job with higher priority. Appendix A presents more details about existing scheduling algorithms and how the job time estimation influence the order in which jobs are executed. In particular it shows that the popular backfilling policy offers good balance between increasing system utilization and maintaining scheduler simplicity.

The key for increasing system utilization is the availability of good resource consumption estimate [50][132]. Unfortunately, it is difficult to estimate accurately the time required for an application to run for various reasons. First, application performance varies from system to system and in practice computing centers are composed of resources from several generations. Even when resources are identical, upgrades of hardware and software are inevitable, rendering previous knowledge obsolete. Second, there are several application parameters that influence the performance of an application. While an experienced user may have previous knowledge to provide a good estimate, keeping track of the variations in performance according to parameters chosen is time consuming. Third, application parallelization is hardly ideal and discovering the percentage of an application that is sequential requires extensive experimentation, adding one more degree of difficulty to estimate runtime when jobs are parallel. All these intricacies lead to systems with highly inaccurate predictions [94][132], a problem that has been tackled for a long time.

Background and Related Work

Machine-Learning Algorithms

Predicting application resource consumption, such as the amounts of CPU, memory, disk and network required by an application and the lengths of time during which the resources are occupied, can be viewed as a supervised machine learning problem. The system needs to learn a concept based on a collection of historical data of *n* previous runs of the application. Each

previous observation is used as training data, providing a set of *m* attributes and the actual outcome: $y = f(a_1, a_2, \ldots, a_m)$. Several parametric (e.g., linear regression, and polynomial regression) and non-parametric (e.g., k-nn, locally weighted linear regression, and decision trees) solutions exist in the field of machine learning, some of which are presented in [5][38][141]. Parametric methods define a hypothesis space and a loss function. The training data is used to model the problem at hand – the parameters of the model are extracted by minimizing the loss function. After the parameters are extracted there is no need for the system to store the training data and predictions can frequently be computed fast, i.e., in O(1) time. On the other hand, non-parametric methods use the actual data as the model, usually presenting prediction computation time that linearly grows with the size of the training data, i.e., takes O(n) time. The advantage in this case is that adding more training data into the learning process is trivial since the next estimate calculation simply takes the new data into account. ML algorithms considered by previous work and evaluated in this chapter, are introduced next.

k-nearest neighbor (k-nn) algorithm: given a query point, the k nearest training data points are considered when computing the prediction. The predicted value can be calculated as the weighted average of selected points. When non-identical weights are used, the weight function usually decreases as the distance from the query point to the training data increases with the goal of minimizing the influence of the distant neighbors on the prediction. While the Euclidean distance is often selected, other distance functions such as Manhattan or Hamming may better represent the gap between data points in certain scenarios. One of the challenges of this algorithm is to find the ideal value for *k*, which is dependent on the distribution of the training data. A high value for *k* can reduce the influence of noise, but in regions scarce in data, the prediction can be highly influenced by distant points. In [119], the best value for *k* is

searched applying a genetic algorithm. *Locally weighted polynomial regression (LWPR)* is similar to k-nn algorithm, in that it considers data close to the query point for the prediction according to a weight function and its kernel bandwidth. Instead of generating the prediction through averaging, the algorithm attempts to fit the data with a polynomial (e.g., a first order polynomial $y = a_1 x + a_0$). The advantage of this method is that predictions can better track data that is not well represented by a plateau, avoiding the rough edges of the k-nn algorithm. It has been demonstrated that the computation demands of both algorithms grows with the size of the knowledge-base, with an additional computation demand for LWPR, but methods combining caching and filtering [72] or kd-trees and approximate weight functions [93] have shown to reduce the computation time manyfold. Previous work [72] has shown that 1-nn and 3-nn with Gaussian weighted averaging can produce lower error rates when predicting the runtime of a short-running scientific application than a locally weighted linear regression; the opposite result applies for a randomly generated dataset.

Linear Regression (LR) algorithm: simple algorithm that finds a linear hyperplane (i.e., $y = a_m x_m + ... + a_2 x_2 + a_1 x_1 + a_0$, where m is the number of attributes) with minimum error in relation to existing data points, defined as the sum of Euclidean distance of each data point to the hyperplane. Once the linear model is defined, prediction takes O(1) time. Linear regression was applied in [3] for predicting execution time of an application in a workflow, and it is also internally used by LWPR and SVM algorithms.

Decision Tree/Table (DT) algorithm: a divide-and-conquer strategy that utilizes a table or tree structure to separate data according to their characteristics. The nodes of the tree (fields of the table) store rules that determine how their children have been split and the leaves (cells) store either the actual outcome, a function that determines the outcome, or the actual data from where

the prediction can be made. This method provides good computational scalability, can handle data that are dispersed in input space well, and its result is reasonably easy to interpret. However, correlations between input characteristics are not well captured since nodes usually split their children according to a single characteristic. The C4.5 classification tree was applied in [108] for predicting execution times in a distributed system that has a centralized scheduler with backfilling, using ranges of execution time as classes. Although implemented using the so called templates that resemble a table rather than a tree graph, [50][94][96][118][132] all used job characteristics to select a subset of the data points on which a statistical function is applied. The choices for the statistical function included the mean [96][118][132], the mean plus 1.96 standard deviations [50], mean plus 1.5 standard deviations [94], linear regression of a single characteristic x ($y = a_1 x + a_0$) [50][96], inverse regression of a single characteristic *x* ($y = x/a_1 + a_0$) [118], and logarithmic regression of a single characteristic x ($y = a_1 \log x + a_0$) [96]. Using the well known space-shared system workloads from the Parallel Workload Archive [43] as the dataset, the mean function has been shown to provide smaller error rates when compared to more complex regressions for the set of characteristics chosen for splitting the training data [96][3].

Artificial Neural Network (ANN): an interconnected group of functions that emulates a biological neural system and can collectively perform complex tasks. During the training phase, an ANN changes its structure based on external or internal information that flows through the network. *Radial Basis Function network (RBFn)* is a feedforward ANN, typically with three layers: input, hidden and output. A variable number of neurons form the key hidden layer, where Euclidean distance from the center of each neuron to the test case is computed and the RBF

[3] The Parallel Workload Archive is a collection of traces; they were not used in this study because they lack application and system performance information needed to evaluate the algorithms under consideration.

function applied. RBFn is very similar to k-nn, except for the fact that RBFn is a parametric method. The number of neurons in the hidden layer (number of neighbors in k-nn) affects the performance and the computational demands of RBFn. In the extreme case (overfitting), each data point can be the center of a neuron (single neighbor in k-nn). The use of RBFn was proposed in [37] combined with a Bayesian network for eliminating attributes with low correlation with the outcome.

Support Vector Machine (SVM) algorithm: a kernel method for solving classification [22] and regression [36] problems, especially for scenarios with non-linear learning pattern. The attribute space is transformed by the kernel function into possibly a high-dimension feature space where an optimal linear hyperplane is found by maximizing the margin between the so called support vectors. The advantages of this method include the limited number of parameters to choose, and the ability to handle large numbers of attributes and non-linear scenarios without local minima problems. A comparison of radial basis function neural networks with SVM has been presented in [37], with lower errors for SVM in some scenarios at the cost of higher computational demand.

Time series methods: in scenarios where the data points of the attribute to be predicted exhibit temporal patterns, algorithms that consider only the order of data points have been proposed. The prediction could be as simple as predicting the future value to be the same as the last observed value, to as complex as considering multiple algorithms in parallel, as proposed by the Network Weather Service (NWS) [142] or a probabilistic approach as a Markov-chain. Triple-C [3] presents a scenario where jobs in a workflow present time dependencies, making Markov-chains appropriate for predicting execution time. In this study, the attributes to be predicted are assumed to have low or no ordering dependency and thus this class of solutions is

not considered. However, the idea of choosing between several learners as in NWS is at the core of the PQR algorithm discussed in the next section.

Input Attribute Selection

Choosing how to best represent the scenario through input attributes can be challenging, often requiring domain knowledge. Previous works have suggested to select properties readily available from batch schedulers such as user name, queue name, number of CPUs, executable name and arguments [50][118], or to select only properties specific to the application [72]. The input attributes have been represented in continuous form [72], in discrete form [118], and in both forms [50]. Continuous attributes have also been normalized with some function [72].

To automatically choose the best set of attributes for predicting job runtime, techniques such as rough sets and a genetic algorithm has been proposed. In [77], the characteristics that are not relevant for the prediction are eliminated using a backward-chaining process based on rough sets theory. In [118], a genetic algorithm is applied to reduce the template search space. The genetic algorithm considers sets of templates as individuals. When forming a new generation, crossover, mutation and elitism are applied. Crossover randomly selects pairs of individuals, giving higher probability for individuals with good fitness to be selected. Once selected, each pair exchanges part of their templates, and bits that represent characteristics are randomly mutated. Elitism guarantees the survival of two best-fit individuals into the next generation without mutation. When compared to the greedy algorithm that searches through all possible combinations, the genetic algorithm has been shown to reach similar or smaller error rate with less computation.

In this work, both resource and application attributes are considered, and the impact of attribute selection on resource usage prediction is evaluated with different combinations of attribute inputs. This choice is based on two facts: (a) application resource consumption depends

heavily on the application inputs and on the target infrastructure, and (b) different cloud providers with various generations of clusters can be selected while computing on the sky. While previous studies focused on a homogeneous resource pool, this work collected resource performance information based on a set of benchmarks.

Predicting Query Runtime Regression (PQR2)

PQR [62] is an algorithm that generates a binary tree that can combine a variety of classifiers. Each node of the tree is represented by a 2-class classifier that is chosen from a pool of classification algorithms according to its accuracy, and the leaves of the tree correspond to ranges of the attribute to be predicted and thus present a coarse granularity of output. PQR has been proposed in the context of predicting execution time of database management systems queries, where cost models are often not good predictors [86] and analytical models are complex and difficult to create [62], yet only coarse classification of queries are necessary.

The PQR tree T_s is defined as a binary decision tree such that:

- Every node u has a 2-class classifier f_u that has been trained on examples E_u
- The classifier f_u decides for each new prediction request r, if r should go to the branch $[t_{ua}, \sigma)$ or $[\sigma, t_{ub}]$, where $t_{ua} < \sigma < t_{ub}$
- Every node u of T_s has an associated accuracy measured as percentage of correct predictions made by f_u on the example set E_u
- Every node and leaf of the tree corresponds to a range of the attribute to be predicted (class)

The algorithm to discretize the numerical attribute into two classes when creating a node consists of ordering all *m* training target attribute values ($a_i, i = 1, \dots, m$) and finding the *k* largest gaps δ defined by $\delta_i = \frac{a_{i+1} - a_i}{a_i}, i = 1, \dots, m-1$ that can become potential split locations (σ). The best combination of classifier and split location with respect to accuracy determines the two attribute ranges.

The procedure to create a PQR tree consists of the recursive application of node splitting procedure until one of the stop conditions is met. Pseudocodes for node splitting and tree construction procedures are presented in Figure 4-2. Once the PQR tree is constructed, each new prediction request that enters the system traverses the tree (models created in each node are responsible for classifying the request as one of the two child ranges) until a leaf is reached. The leaf indicates the predicted range.

Node Splitting Procedure (split):

- Sort the target attribute $a_1,..,a_m$ of m training examples in ascending order, skipping n_{skip} percentage values from the beginning and end of this list L
- Compute gaps δ in L, given by the formula $\delta_{i+1} = \frac{a_{i+1} - a_i}{a_i}$
- Obtain a set Δ of n_{gap} largest δ_i
- For each $\delta_i \in \Delta$, partition the training set into two subsets according to δ_i
- For each classifier $f_u \in F$, train f_u and predict the range for each example, computing f_u accuracy (percentage of correct predictions made by f_u)
- Choose the combination of classifier f_u and split location σ that gives the best accuracy

PQR tree construction procedure (build):

- [u, E_{ua}, E_{ub}] = split(E_u)
- Continuea = true, Continueb = true
- If $Min_{accuracy}$ > accuracy(u) return
- If $Min_{example}$ > count(E_{au}) Continuea = false, If $Min_{example}$ > count(E_{bu}) Continueb = false
- If $Min_{intervalsize}$ > range(E_{au}) Continuea = false, If $Min_{intervalsize}$ > range(E_{bu}) Continueb = false
- If 2 > count((1 – 2 * n_{skip}) * E_{au}) Continuea = false
- If 2 > count((1 – 2 * n_{skip}) * E_{bu}) Continueb = false
- If Continuea build(E_{au})
- If Continueb build(E_{bu})

Figure 4-2. Pseudocode for creating a PQR tree. The build procedure shows the various parameters for controlling the tree growth, while the split procedure shows how to split the data of a node into two classes and how to choose a binary classifier.

The growth and quality of the tree can be controlled by tuning several parameters of the PQR algorithm that dictate the minimum number of training data in a node, the minimum accuracy required by a classifier, the minimum attribute interval covered by a node, and the

percentage of data points from the extremes of prediction attribute that should not be considered for split. Although many combinations of PQR parameters are possible, in practice, the use of default values suffices to achieve good accuracy.

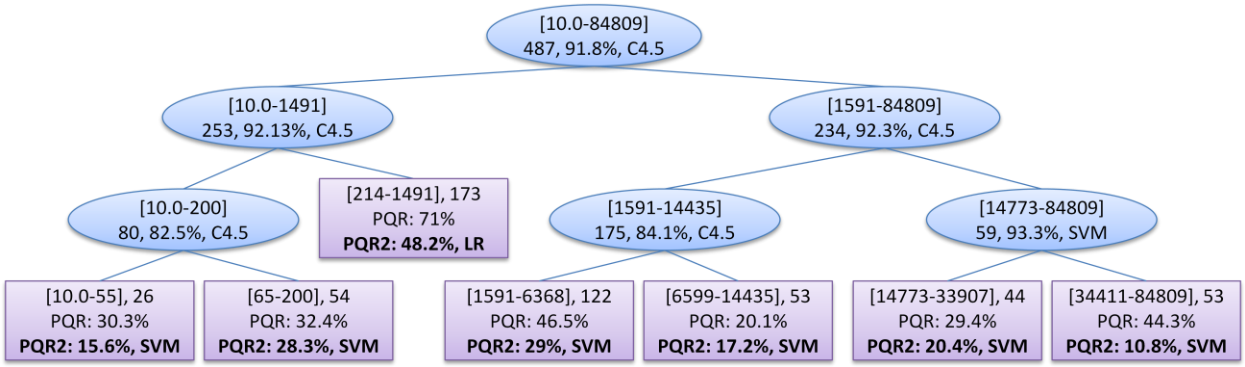


Figure 4-3. Tree generated by PQR and PQR2 algorithms. The nodes of the tree (circles) are common to both methods, while leaves (squares) of PQR2 (boldface) yield lower errors than PQR (normal face). The improvement comes from the ability of selecting the best regression method from a pool, whereas leaf range median is used in PQR. The number between square brackets represents the range of values of the attribute to be predicted, which is followed by the number of historical data points in each node/leaf. The percentage value indicates the accuracy of each classifier (nodes) or the percentage error of each regressor (leaves). The last value indicates the name of each classification (PQR and PQR2) or regression (PQR2) algorithm selected.

In this work, we generalize PQR to the regression problem by training regression algorithms from a pool of known methods with the data on PQR leaves and choosing the best algorithm. Any regression algorithm can be placed on the pool, including different configurations of the same algorithm, but preference for parametric methods should be given when fast predictions are required. Figure 4-3 highlights through an example the differences between a PQR and a PQR2 tree. The nodes of the tree, shown as circles with the name of selected classifier, are produced using the algorithm described in [62]. Each node is responsible for classifying data into two classes, defined by a threshold that is chosen as to maximize the accuracy of the predictor. The values between brackets show the range of attribute values that were observed by each node or leaf during training, followed by the amount of historical

information. The key difference is found at the leaves: instead of outputting classes or the range median, PQR2 generates the best regression model for the available data (LR and SVM in the case of the leaves shown in the figure). The percentage error (displayed in bold) in each subspace is lowered by PQR2, which offers fine-grained prediction through models that adapt better to the characteristics of the data in each leaf.

Experimental Setup

To generate data for this study, a heterogeneous environment consisting of four different hardware resources specified in Table 4-1 were used to run two popular bioinformatics applications, namely Basic Local Alignment Search Tool (BLAST) [6] and Randomized Axelerated Maximum Likelihood (RAxML) [124].

Table 4-1. Specification of resources used to generate scenarios for prediction study.

Cluster	c1	c2_512M	c2_3.5G*	c3*	c4
CPU/node	2-way Pentium III (2 Tualatin cores)	2-way Xeon (2 Prestonia cores)	2-way Xeon (2 Prestonia cores)	2-way Xeon (8 Clovertown cores)	16-way Xeon (64 Tigerton cores)
CPU clock	1.4 GHz	2.4 GHz	2.4 GHz	2.33 GHz	2.93 GHz
CPU cache♢	512 KiB/core	512 KiB/core	512 KiB/core	4 MiB/2-cores	4 MiB/2-cores
CPU speed	659 MFlops	879 MFlops	879 MFlops	1436 MFlops	1798 MFlops
Kernel♢♢	2.6.23.8-i686	2.6.18-i686	2.6.18-i686	2.6.22.2-x86_64	2.6.18-x86_64
Xen VMM	-	3.1.0	3.1.0	-	-
Memory♢	2 GiB	512 MiB	3.5 GiB	3.9 GiB	503 GiB
Memory speed♢	437 MiB/s	1932 MiB/s	1932 MiB/s	3468 MiB/s	1257 MiB/s
Disk read speed	65.5 MB/s	37.8 MB/s	37.8 MB/s	60.2 MB/s	240.9 MB/s

♢Following the IEC 80000-13:2008 standard, the prefixes KiB (kilobinary bytes), MiB (Megabinary bytes) and GiB (Gigabinary bytes) are used to represent multiples of 2^{10}, 2^{20}, and 2^{30} respectively.
♢♢ i686 suffix indicates a 32-bit kernel while x86_64 suffix indicates a 64-bit kernel

BLAST version 2.2.18 (either 32-bit or 64-bit depending on the resource) was executed against the non-redundant (NR) protein sequence database from NCBI split into 1 fragment (total

of 3.5 GB of data). Given an input sequence, BLAST searches a database for similar sequences and calculates the best alignment of the matched sequences. Single nucleotide sequences of varying lengths served as input (see sequence length distribution in Figure 4-4) in this study.

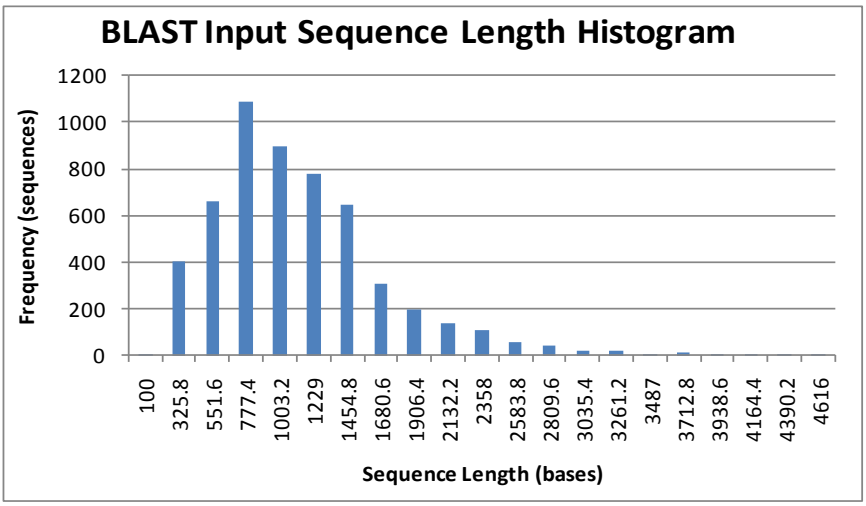


Figure 4-4. Histogram of input sequence length for BLAST. The length of input sequences is one of the important features influencing the execution time of BLAST.

In addition to running BLAST on all different resources on local disk, BLAST was also run placing the database on two NFS servers with different hardware (the * on Table 4-1 indicates the physical resources used as NFS servers; the disk performance, according to the number of clients, has been used to model the load on the server and is shown on Table 4-2). The purpose of including runs with the database placed on remote servers is to include scenarios where applications space-share local resources, but time-share other resources such as network or a file system, a typical scenario on clouds that serve virtual machine images or user's data from a shared server. BLAST runs resulted in a collection of 6592 data points. In order to provide insight about the characteristics of this dataset, partial data is plotted on Figure 4-5 (top) to show the existence of a linear relationship between BLAST execution time and input sequence length that is different for each of the cluster resources used. This linearity is affected when the

database is placed on a file server that is concurrently accessed by different number of clients, especially for shorter input sequences (Figure 4-5 bottom).

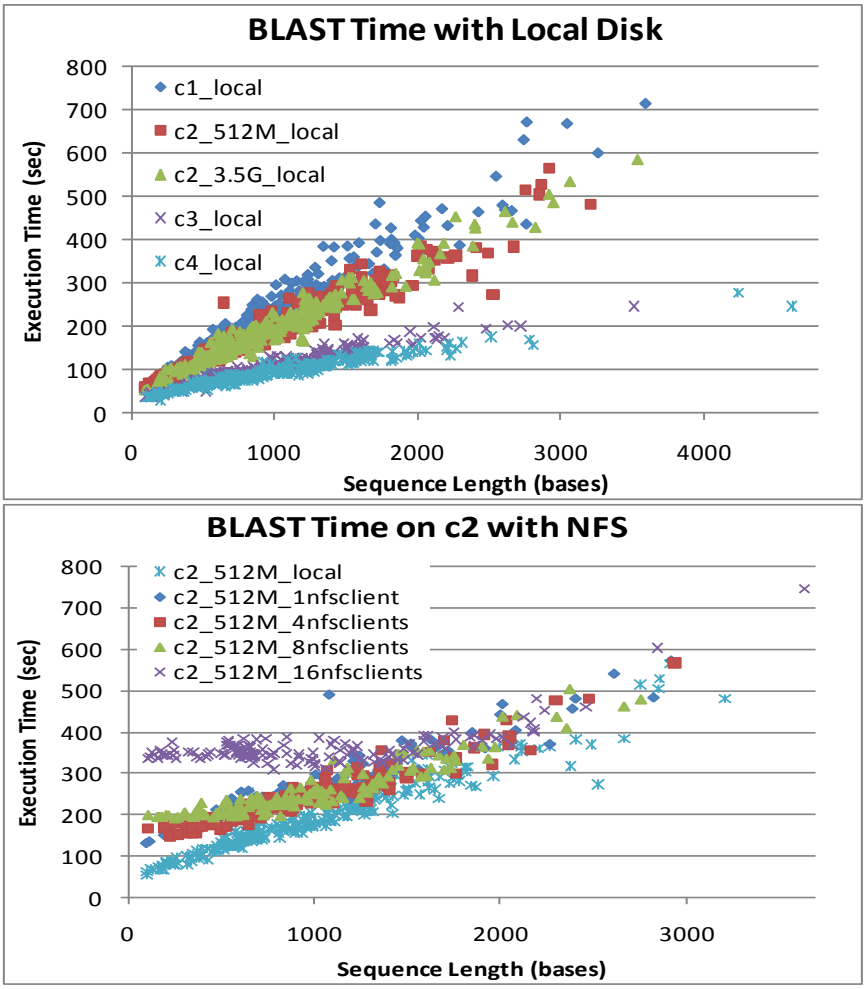


Figure 4-5. Execution time of BLAST according to individual sequence length. An approximately linear relationship between time and input sequence length that is distinct for each different resource can be observed (top). When data is placed on a NFS server, varying the number of concurrent clients on a particular resource (c2) affects the linearity, especially for short sequences (bottom).

Table 4-2. Disk performance when varying the number of concurrent clients.

NFS server	1 client	4 clients	8 clients	16 clients
c2_3.5G	20.6 MiB/s	13.5 MiB/s	9.3 MiB/s	5.8 MiB/s
c3	28 MB/s	23 MB/s	13 MB/s	7 MB/s

RAxML version 7.0.4, a software application for constructing phylogenetic trees using maximum likelihood analysis, was executed for 35 nucleotide datasets combining five different

taxa sizes (50, 100, 150, 200 and 250), and seven different sequence lengths (5000, 10000, 15000, 20000, 25000, 40000, and 50000). Runs for different numbers of threads, up to the number of cores in each resource, were also performed, resulting in a dataset with 487 data points.

RAxML is not only computationally intensive, but also memory intensive. Non-linearity of execution time with respect to resources and application characteristics occur in two situations in this dataset: as number of threads increase and as dataset increase. For inputs with 250 taxa with different lengths, the execution time (in logarithimic scale) in Figure 4-6 shows that single-threaded runs on c3 are faster than on c4 (which has slower memory). This fact slowly changes as the number of threads is increased. Similar trends are observed for other datasets and resources (not shown due to the amount of data; full dataset can be found in [85]). Furthermore, although larger inputs are expected to require more execution time, in some cases, larger inputs have shorter execution time. For the experiments shown in the next section, attributes of applications (sequence length, taxa size, and taxa size multiplied by sequence length) and resources (cluster name, CPU clock, amount of cache, and amount of memory) were selected. However, as information such as CPU clock does not reflect well the processing capability of each resource, the following benchmarks were used to better characterize the performance of each resource (included in Table 4-1 and Table 4-2):

- A simple matrix multiplication application was used to extract the number of floating point operations performed in a fixed amount of time (CPU speed).
- A cache benchmark application was used to collect the performance of a read-modify-write operation (memory speed).
- The Linux ‘dd’ command was used to read a 1 GiB file from each local disk and from each NFS server varying the number of clients (disk read speed). Read benchmark was preferred in this study (instead of a write benchmark) due to the large database read by BLAST. In other scenarios, write performance could also be used.

The overall average and standard deviation of BLAST and RAxML datasets used in the experiments and evaluated in the next section are shown in Table 4-3.

Table 4-3. Overall Characteristics of Learning Datasets.

Dataset	Mean	Standard Deviation
BLAST output	50025 bytes	129743 bytes
BLAST time	208.4 secs	97.7 secs
RAxML RSS	110442 bytes	96077 bytes
RAxML time	5720 secs	10702 secs

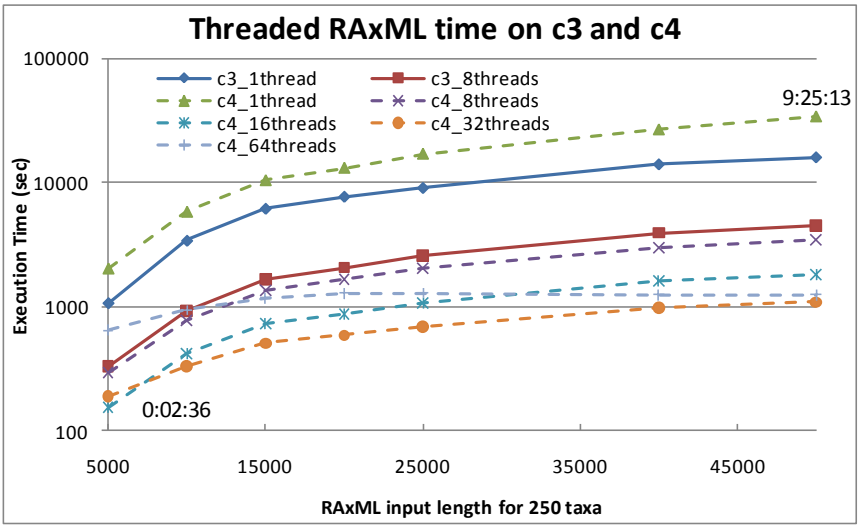


Figure 4-6. Execution time of RAxML on c3 and c4 resources with different number of threads. Input datasets with 250 taxa and lengths varying from 5,000 to 50,000 were used. The numbers next to extreme points indicate the minimum and maximum execution time (hh:mm:ss). Single threaded execution on c3 is shorter than on c4, but this relationship is inverted when running RAxML with 8 threads.

Experimental Results and Evaluation

Application resource usage prediction is evaluated in this section for nine different configurations of machine learning algorithms (see Table 4-4), including the new PQR2. All experiments were performed using Weka version 3.0.7, a Java-based open source collection of ML algorithms and statistical tools with graphical user interface for visualizing data, processing information, and carrying data mining experiments. PQR and PQR2 were newly implemented in Weka, extending defined interfaces, making use of data manipulation utilities, and reusing existing ML algorithms such as Naïve Bayes [67], C4.5, SVM and LR.

Three attributes often considered for prediction are the application execution time, the amount of resident memory (Resident Set Size or RSS) required, and the size of output produced. Evaluating these attributes for BLAST and RAxML datasets described in the previous section, produces six scenarios. However, due to the constant nature of BLAST memory usage (defined by the NR database size) and RAxML output for the runs performed, evaluations presented in this section did not include these two scenarios. For the remaining four scenarios, 10 iterations of 10-fold cross-validation were performed for each algorithm and scenario to obtain results that are statistically relevant, resulting in 100 evaluations of each experiment.

Table 4-4. ML Algorithms Investigated.

Abbreviation	ML	Configuration
Knn1nInv	k-nn	Single neighbor
Knn30nInv	k-nn	30 neighbors with weight inversely proportional to the distance
Knn30nNow	k-nn	30 neighbors without distance weight
LinearGdy	LR	Greedy selection of attributes
Dectable	DT	Forward best first attribute selection considering 5 non-improving nodes
RBFn50c01s	RBFn	50 clusters (neurons), 0.1 minimum standard deviation
RBFn200c01s	RBFn	200 clusters, 0.1 minimum standard deviation
Pqr1_75	PQR	0.75 minimum accuracy threshold, nskip=25%, ngap=5, minexamples=30, minintervalsize=5, considering Naïve Bayes, C4.5 and SVM as node classifiers
Pqr2_75	PQR2	Same as Pqr1_75, with range median, average, LR and SVM as leaf regressors
SmoPoly1	SVM	linear kernel
SmoPoly2	SVM	polynomial kernel of degree 2

The Receiver Operating Characteristic (ROC) curve is a method for visually comparing accuracy of classification algorithms – it was generalized for regression algorithms as Regression Error Characteristic Curves (REC) [20]. The REC curve displays the error tolerance in the horizontal axis and the accuracy of an algorithm in the vertical axis, with accuracy defined as the percentage of predictions below the error tolerance threshold. The area over the curve represents the expected error for a regression model. When the curve of a certain algorithm A always

appears above the curve of another algorithm B, A is the preferred algorithm and A is said to dominate B. In this study, the percentage error ($\frac{|p_i - a_i|}{a_i}$, where p_i is the predicted value and a_i is the actual value for a test case i) was chosen as the error metric displayed in the REC curves. Thus, the area over the REC curve is close to the mean percentage error (MPE). Other possible performance metrics are shown in Table 4-5.

Table 4-5. Performance metrics for numerical predictions.

Performance measure	Formula
Mean-Squared Error (MSE)	$\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}$
Root Mean-Squared Error (RMSE)	$\sqrt{MSE}$
Mean Absolute Error (MAE)	$\frac{\sum_{i=1}^{n}\lvert p_i - a_i\rvert}{n}$
Mean-Squared Percentage Error (MSPE)	$\frac{\sum_{i=1}^{n}(\frac{p_i - a_i}{a_i})^2}{n}$
Root Mean-Squared Percentage Error (RMSPE)	$\sqrt{MSPE}$
Mean Percentage Error (MPE)	$\frac{\sum_{i=1}^{n}\frac{\lvert p_i - a_i\rvert}{a_i}}{n}$
Relative Squared Error (RSE)	$\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{\sum_{i=1}^{n}(a_i - \bar{a})^2}$, where $\bar{a} = \frac{\sum_{i=1}^{n} a_i}{n}$
Root Relative Squared Error (RRSE)	$\sqrt{RSE}$
Relative Absolute Error (RAE)	$\frac{\sum_{i=1}^{n}\lvert p_i - a_i\rvert}{\sum_{i=1}^{n}\lvert a_i - \bar{a}\rvert}$
Relative Squared Percentage Error (RSPE)	$\frac{\sum_{i=1}^{n}\left(\frac{p_i - a_i}{a_i}\right)^2}{\sum_{i=1}^{n}\left(\frac{a_i - \bar{a}}{a_i}\right)^2}$,
Root Relative Squared Percentage Error (RRSPE)	$\sqrt{RSPE}$
Relative Percentage Error (RPE)	$\frac{\sum_{i=1}^{n}\left\lvert\frac{p_i - a_i}{a_i}\right\rvert}{\sum_{i=1}^{n}\left\lvert\frac{a_i - \bar{a}}{a_i}\right\rvert}$
Number of underestimates	$\sum_{i=1}^{n} 1(a_i - p_i)$, where $1(x) = \begin{cases} 0, \text{if } x < 0 \\ 1, \text{if } x \geq 0 \end{cases}$
Number of overestimates	$\sum_{i=1}^{n} 1(p_i - a_i)$

For a test case of n values, p are predicted values and a are the actual values. Formulas partially provided by [141].

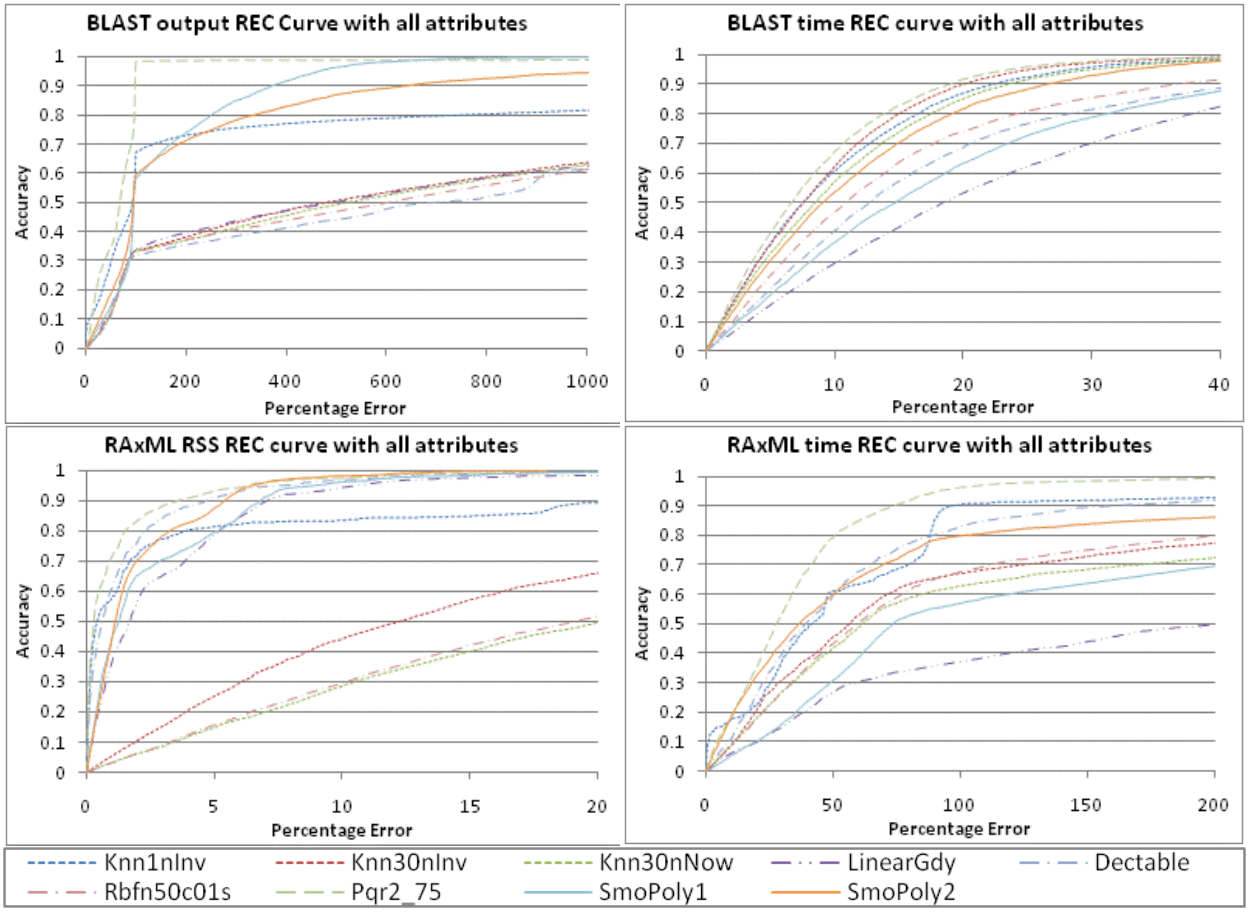


Figure 4-7. REC curves predicting resource requirements by BLAST and RAxML. Performance of various machine learning algorithms predicting disk space (top left) and execution time (top right) for BLAST, and resident memory (bottom left) and execution time (bottom right) for RAxML is shown when taking all attributes into consideration. The area over the curve represents the mean percentage error, which is presented in Figure 4-8 and Figure 4-10.

Evaluation of ML algorithms is performed by answering four questions.

Question 1: Which ML algorithm offers the best accuracy?

To evaluate accuracy, REC curve for each ML algorithm was generated for all four scenarios, when all system and application attributes are included in the training dataset (Figure 4-7). The fact that PQR2 dominates other studied algorithms in all scenarios (top left most curve in all graphs), shows that it can adapt better to different scenarios, where the predicted resource usage presents linear (RAxML RSS in the presence of input size information) and non-linear behavior (other scenarios) as a function of system and application characteristics. This adaptation

is due to the fact that different classification and regression models are created for different regions of the data space, at the cost of additional computation during training phase. The figure also shows the difficulty in finding an optimal parameter for certain ML algorithms (e.g., different number of neighbors in k-nn offer best performance in different scenarios).

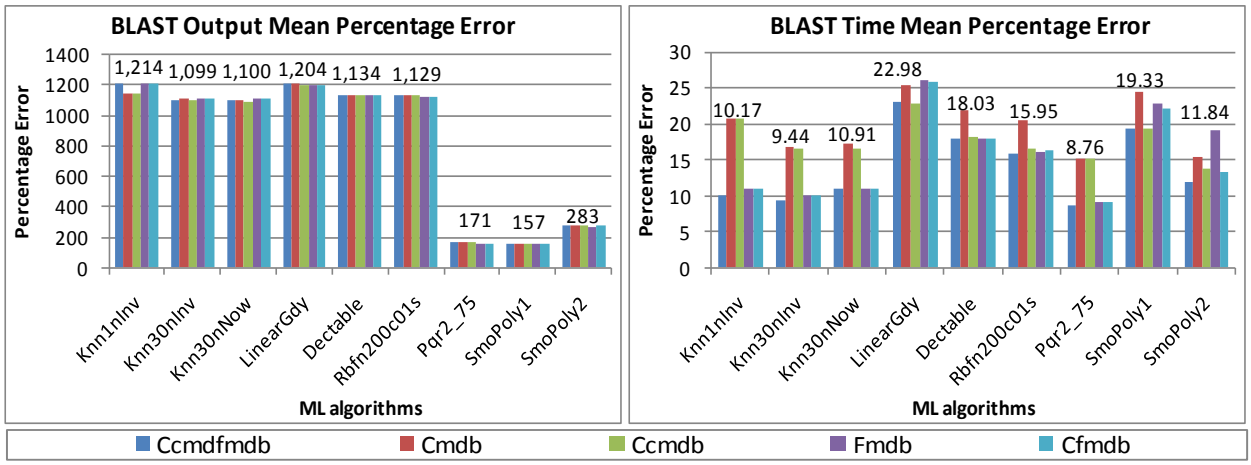


Figure 4-8. Average percentage errors in predictions for BLAST resource requirements for various ML algorithms. High percentage error show the difficulty in predicting disk space (left) required by BLAST. On the other hand, predicting execution time (right) presents low error values. Values in the graph correspond to the configuration using all available system and application attributes.

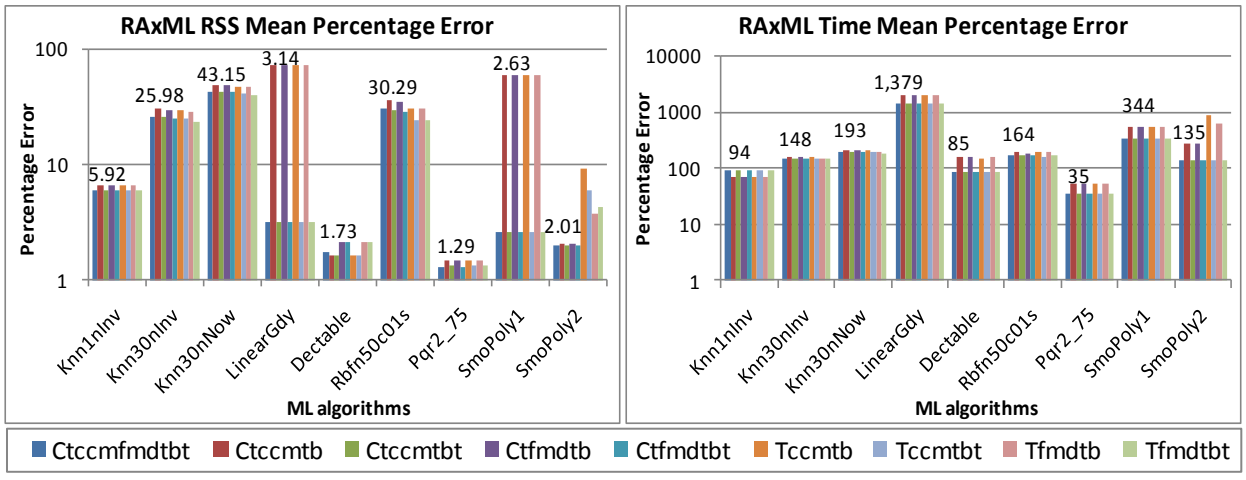


Figure 4-9. Average percentage errors in predictions for RAxML resource requirements for various ML algorithms. High variation on percentage error, when comparing different ML algorithms and different selection of attributes, is observed for predicting both resident memory (left) and execution time (right) required by RAxML (error axis in logarithmic scale).

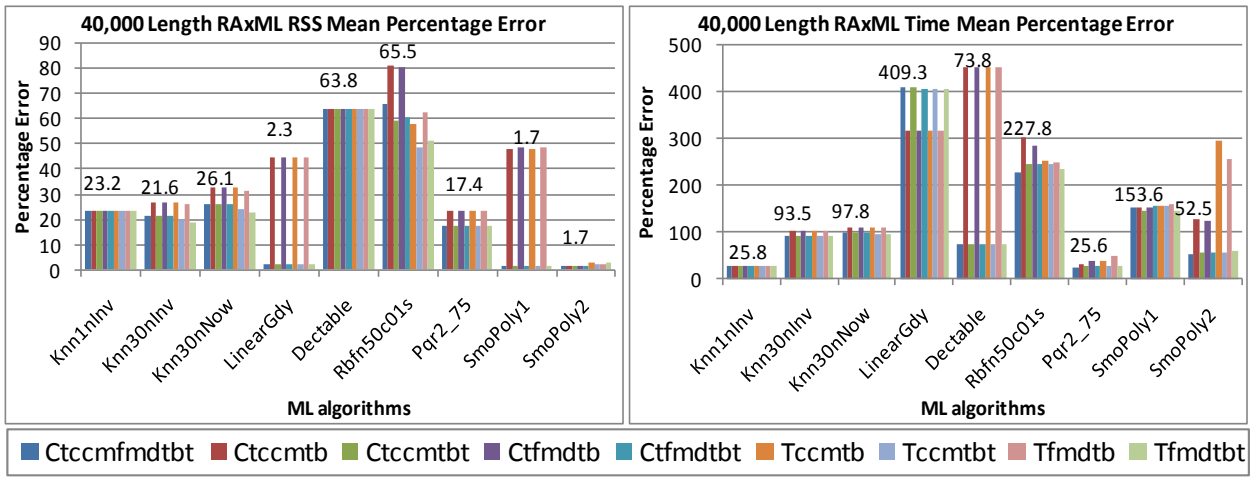


Figure 4-10. Average percentage errors in the absence of similar data for various machine learning algorithms. Results show performance predicting resident memory (left) and execution time (right) required by RAxML for all cases with 40,000-long sequences, using other data points as training data.

SVM and k-nn algorithms are in general the next best algorithms. In particular, since the BLAST dataset contains a large number of data points, the REC curves show that increasing the number of neighbors to 30, increases the accuracy of k-nn, but this effect is opposite for the RAxML dataset. In fact, due to the limited number of data points, increasing the number of neighbors in the RAxML scenario is extremely harmful, even more so when no weight is used. This result indicates the difficulty in finding an ideal number of neighbors, especially when the training information presents varying density of points across the data space. With respect to SVM, the graphs show that the polynomial kernel of degree 2 can adapt better to the non-linearity of data than the linear kernel, especially in the RAxML scenarios, without requiring ideal algorithm parameters to be found.

Similar to k-nn, RBFn requires the number of neurons to be specified, a configuration that dramatically impacts the prediction accuracy. Empirical tests showed that the ideal number of neurons for BLAST scenarios was 200, while 50 neurons were the best configuration for RAxML. These configurations were used in the comparative graphs.

Question 2: Which attributes should be included in the training dataset?

Previous works [37][50][77][96][118] have proposed and studied methods for systematically selecting attributes independently from the learning algorithm. In this work, the impact of different sets of attributes on the accuracy of prediction is evaluated. Figure 4-8 and Figure 4-9 show the MPE for BLAST and RAxML respectively, for different sets of attributes. The number that appears above each group of vertical bars is the MPE of the case including all attributes (leftmost bar).

In the BLAST scenario, the attributes included are: cluster name, CPU clock, amount of memory, location of data (nominal value indicating the name of the resource hosting the data), CPU speed (Flops), memory speed, disk speed and number of bases in a sequence. Since the output size is not dependent on any of the system attributes, the selection of attributes has almost no impact on the accuracy of BLAST output prediction (Figure 4-8 left). Thus, maintaining them in the dataset is not detrimental to the performance of ML algorithms. On the other hand, the addition of attributes, by collecting either more detailed information about the application being executed or more system benchmark and monitoring data, can highly impact the prediction accuracy. Observing k-nn, decision table and PQR2 results, the system performance attributes offer better prediction (Fmdb and Cfmd cases) than simply using system specification attributes (Cmdb and Ccmdb). This fact makes the case for clusters, grids, and clouds middleware to provide the complete set of information to users and learning systems.

In the RAxML case, cluster name, number of threads, CPU clock, amount of cache, amount of memory, CPU speed (Flops), memory speed, disk speed, taxa size, number of bases in a sequence, and input size (taxa size multiplied by number of bases) were considered. The graphs with MPE, displayed in logarithmic scale (Figure 4-9), show PQR2 as the best algorithm for

predicting memory and execution time requirements. Due to the linear relationship between the input size and RSS required by RAxML ($\mathrm{mem}(t, b) = (t - 2) * b * 32$, where t is the taxa size, and b is the number of bases), the input size is the attribute with the most impact on RSS prediction (cases ending with "t"), especially for the linear models (LR and SVM with linear kernel). As seen in the previous section, the execution time of RAxML is non-linear with respect to input size, number of threads and resources, leading in general to a high MPE. Still, PQR2 can adapt better to this situation, followed by decision table, k-nn, and SVM with polynomial kernel of degree 2. The decision table yields good prediction in this case due to the clusterized distribution of data points. The input size attribute continues to impact accuracy of time prediction more than the other attributes, including system performance attributes, making the case for Software-as-a-Service providers and job management systems to automatically collect more detailed application-specific information.

Question 3: Which ML algorithm provides better accuracy when dealing with training datasets with low coverage?

To evaluate this situation, instead of separating training and test data in a random fashion, all data points of RAxML executions with 40,000 bases long input are used as test data and omitted from the training set. The MPE obtained for each combination of ML algorithm and attribute set show that for RSS prediction, the linear models can capture very well the overall model when the input size attribute is present, as expected (Figure 4-10 left). However, this outcome does not outweigh the benefits of other solutions in the more general scenario. The split of data in PQR2 diminishes the performance in this condition when compared to methods that produce a global model, but PQR2 still models better the general trend than the decision table and the k-nn, which always select the 50,000 data points as the closest neighbors. It can also be

observed that the clustered nature of this dataset makes RBFn perform poorly since regions without data points are not included in the model.

For time prediction, the algorithms that can deal with non-linear execution times, PQR2 and k-nn, provide the best models even though no data points for 40,000-long inputs are included in the training set.

Question 4: Does PQR2 offer better accuracy than PQR?

REC curves (Figure 4-11) and the MPE (Table 4-6) show that PQR2 offer considerable improvement for predicting BLAST output, small benefits for predicting RAxML memory and time consumption, and essentially no difference predicting BLAST execution time, when compared to PQR. This is explained by the average number of leaves produced by PQR/PQR2 (Table 4-6). When the tree has a large number of leaves (37 for BLAST time model), the amount and range of data in each leaf, on which PQR2 can work to improve prediction accuracy, are small.

PQR2 models created with the entire dataset selected either the linear regression or SVM regression for modeling the data on the leaves, with a clear tendency to select SVM. This confirms that adding regression algorithms to the leaves is beneficial (PQR2) when compared to simply using the range mean or median (PQR). Considering the selection of classifiers, common to both PQR and PQR2, the resulting models included all classifiers, with strong tendency towards selecting C4.5.

Table 4-6. Comparing PQR and PQR2 with MPE metric.

Dataset	Pqr1_75	Pqr2_75	Leaves
BLAST output	962%	171%	15
BLAST time	8.81%	8.76%	37
RAxML RSS	4.54%	1.29%	24
RAxML time	40.82%	35.30%	24

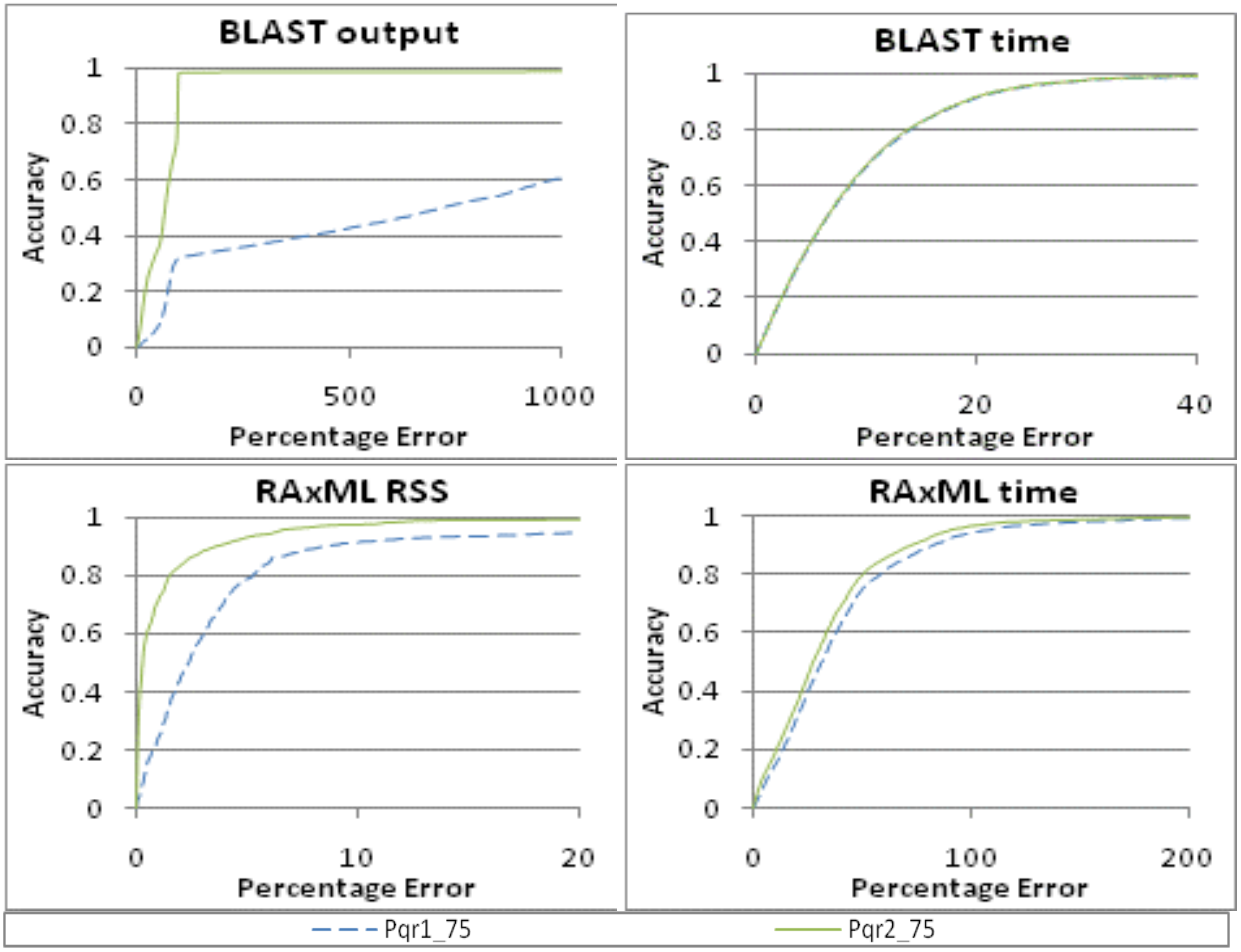


Figure 4-11. REC curves comparing accuracy of PQR and PQR2. Performance predicting BLAST output (top left) and execution time (top right) as well as RAxML memory consumption (bottom left) and execution time (bottom right) indicate better accuracy of PQR2.

Summary

The research reported in this chapter considered the problem of accurately predicting application resource usage, reviewed and discussed several noteworthy machine learning algorithms considered by previous work, proposed PQR2, an extension of an existing classification tree algorithm, and compared all solutions (including the new PQR2 algorithm) under several conditions. Both PQR and PQR2 algorithms were implemented for the Weka framework and made publicly available. Experiments predicting execution time, memory and disk requirements for two popular bioinformatics applications, BLAST and RAxML, were

performed on a heterogeneous environment. Overall, PQR2 exhibited better accuracy when compared to other algorithms, due to its ability to better adapt to scenarios with different characteristics (linear and non-linear relationships, high and low density of training data points) by choosing different models for its nodes and leaves.

At a more general level, the two main conclusions from the work reported in this chapter are as follows:

- The scenarios requiring application resource prediction present a diverse behavior, making different algorithms perform better in different situations. The use of methods that can adapt to these situations by considering different configurations and algorithms is key for improving the quality of the prediction without requiring manual tuning. The resulting algorithms may be computationally more demanding during training, but this is usually not a concern as there is no need to generate a new model very often. Nevertheless, the time and memory consumed by the prediction procedure for all ML algorithms is the subject of future work, with room for optimizations. Roughly, using the largest dataset (BLAST), PQR2 required a few minutes to create the model and a few milliseconds to produce a single prediction, indicating practicality of PQR2 for production deployments. PQR2 proved to be the best solution for BLAST and RAxML and should be considered as candidate solution for other applications.

- Attributes can have high impact on the performance of the learning algorithms. The use of system performance attributes showed to be relevant for execution time prediction whereas application specific attributes were pertinent for all scenarios. This work makes the case for including as many attributes as available, while letting the algorithms analyze the relevance of the attributes when necessary. For cloud and grid computing scenarios, where resources are outsourced, the provision of this information to its users (or services acting on behalf of the users) through the use of benchmarks and runtime monitoring, especially of shared resources, can bring several benefits. Although this information is not readily available on a per application run basis, we expect it to become available in the near future (Amazon CloudWatch is one such example limited to a virtual machine instance). Improved prediction can result in better system utilization [50], can avoid application abortion in system that enforce accurate resource reservation, as well as significant cost savings when choosing the appropriate pay-as-you-go resource.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

Contributions

As cloud computing becomes more commonplace, methods to make the best use of computing and human resources in this new paradigm are essential. Significant advantages of cloud computing for consumers are the absence of up-front costs of resource acquisition and installation, and the ability to rapidly and dynamically size the amount of resources allocated for an application. IaaS and PaaS providers can exploit economies of scale while SaaS providers can offer additional higher-level services to consumers for a cost. The thesis of this work is that efficient use of cloud computing systems can be achieved by automating processes currently carried out by end users and developers.

In particular, challenges addressed by this work include: (a) the need to automatically recast existing applications as services to facilitate interoperability and integration of applications to create new complex services, (b) the need of a roadmap for rapidly deploying scientific application on the cloud by combining technologies that overcome the complexity of dealing with multiple heterogeneous clouds distributed across a WAN and development of efficient solutions for the parallel execution of unmodified application, and (c) the need to accurately predict application resource consumption for several purposes including the goal of increasing scheduler performance and cloud cost estimation accuracy.

The main contributions and conclusions of this work follows.

- Classification of applications that led to the development of a tool called CLAWS targeting WS development and deployment environments. When used with Java, Tomcat and Axis, CLAWS provides a framework for rapidly wrapping text-oriented applications as Web Services for both interactive and stateful applications, independently of the operating system required to run the application. A framework like CLAWS is expected to increase the number and variety of SaaS offerings by speeding up the process of enabling applications over the Internet.

- Evaluation of the effectiveness of CLAWS on Windows and Linux, demonstrating the flexibility of the approach handling applications with different characteristics. During the Transnational Digital Government project, CLAWS successfully and rapidly enabled and deployed dialogue and translation applications as Web Services in Belize and the Dominican Republic, being a key component in the integration of a transnational border control information sharing system.

- CloudBLAST approach facilitates the scaling out of embarrassingly parallel applications using machines from multiple IaaS clouds that are inherently disconnected and heterogeneous with respect to performance. Scalable management of VMs, connectivity, deployment, and performance was achieved by combining Nimbus toolkit, ViNe, Hadoop, and skewed distribution of tasks.

- Experiments with a bioinformatics application in a real meta-cloud testbed demonstrated insignificant overhead due to the use of machine and network virtualization. It also showed that while both MapReduce and MPI implementations of BLAST scaled increasing the number of resources allocated to solve a problem, a MapReduce implementation such as Hadoop brings with it significant advantages from the perspective of management of failures, data and jobs.

- Behavior of several existing machine-learning algorithms was investigated when applied to the problem of predicting application resource usage using information gathered from two bioinformatics applications executed on a heterogeneous set of resources. While previous studies proposed the use of different algorithms, few provided comparison with other approaches. The comprehensive evaluation of ML algorithms in this study fills this void in a number of scenarios using different combinations of attributes and datasets with low coverage in certain regions.

- A new machine-learning algorithm, PQR2, was proposed consisting of improvements to the PQR algorithm, (a) enabling more classification algorithms to be considered as tree nodes and (b) adding regression methods from a pool to be chosen for providing further estimation local to a range of the PQR tree. Experimental results showed that PQR2 is the best candidate to be deployed in clouds that can collect both application and resource performance information.

Combining CLAWS, CloudBLAST and PQR2

CLAWS, CloudBLAST, and PQR2 (described respectively in Chapters 2, 3, and 4) automate procedures currently carried out by end users. These solutions are also expected to be incorporated by SaaS, PaaS and IaaS providers. A natural advancement is the integration of these solutions (see Figure 1-3) to provide a seamless execution environment for the end user to run unmodified applications. CLAWS enables SaaS providers to automatically generate application-

specific Web services and interfaces. The services are based on command-line scripts that make use of PaaS providers using CloudBLAST. PaaS providers in turn make use of multiple IaaS providers that require estimation of application resource usage provided by PQR2. PQR2 collects application-specific features from SaaS providers and resource-specific features from IaaS provider's SLA and monitoring services. Extensions to CLAWS, CloudBLAST and PQR2 for broadening the applicability of the integrated solution are discussed in detail next.

Extending CLAWS

The wrapping of existing applications as Services offered by CLAWS assumes that applications are to be locally executed, on the same resource running the Service engine. Many projects in grid computing and scientific applications domain have addressed a similar problem, where the typical application being enabled is computationally intensive, requiring the application to be executed on a large number of resources in a non-interactive and stateless fashion. Noteworthy approaches include: SoapLab [113], GridLab's Grid Application Toolkit (GAT) [4], Virtual Application Services (VAS) [84], Generic Application Service (GAP) [112], and Generic Application Factory (GFac) [71]. In grid and cloud scenarios, features such as resource discovery, distribution of work, notification of execution progress, and delegation of credentials become very important. CLAWS does not support these complex features, but it is flexible enough to support a wide range of applications, one of which could be a grid or cloud toolkit client command line that would execute the application in a distributed fashion and return the result.

There are other specific differences between CLAWS and grid solutions (GAP, VAS, and GFac). CLAWS offers a web interface for the users to specify the command-line instead of requiring the specification to be handed in an XML file, a task that not all users are comfortable with, even with the advent of XML editors. The solutions also differ in granularity of command-

line options, configuration definitions and validation features. For example, VAS and GAP can accept specifications of complex command-line options [149] whereas GridLab and GFac only deal with simple command-line arguments. CLAWS handle an intermediate level of complexity in command-line specification, supporting the definition of parameters that can be referenced when constructing not only the command-line, but also the input format, a feature essential for handling interactive applications and absent in other approaches. Incorporating the idea of command-line validation present in GAP and VAS, through specification of input inter-dependencies and input data type, will increase the service security by permitting only specific inputs to be provided to the service. The challenge resides in providing additional flexibility in input specification (e.g., valid attribute range for numeric inputs, inappropriate patterns, inter-relationships between inputs) while keeping a user-friendly interface (e.g., a graphical interface for specifying dependencies, a list of pre-defined rules).

With respect to Service input and output formats, grid solutions allow more detailed description, in the form of complex objects, and files are handled by a separate mechanism or as an additional service. Text type input and output are assumed by CLAWS since the Services are to be located behind customized Web portals with text-only input forms. Extending CLAWS to take advantage of IaaS storage providers for application input and output, in way similar to that offered by file transfers in grid solutions, will provide support for applications with large datasets.

Extending CloudBLAST

CloudBLAST approach offers an efficient solution for distributing the execution of an important type of application – those that are embarrassingly parallel. For the applications that do not fit well the MapReduce abstraction, other computation distribution abstractions such as workflows have been proposed (e.g., Microsoft Dryad [65], and Kepler [81]). These solutions

have in common the idea of representing a large application as a graph (nodes represent computation and edges indicate the flow of information), but differ in several other aspects. The representation of the workflow can be provided in one of many languages (e.g., XML, GUI, and scripting); the graph can be cyclic or acyclic; flow of information can be synchronous or asynchronous using different protocols and formats; control of tasks can be centralized or distributed requiring services or resources to be specified or offering scheduling algorithms; diverse set of remote execution is supported; and different types of fault-tolerance may be provided. For applications that fit well the MapReduce or other abstractions, but present intensive data manipulation operations such as joins and filters, higher-level abstractions that increase productivity exist (e.g., Pig, and DryadLINQ).

A broader range of applications can be supported by CloudBLAST extending it to work not only with MapReduce but also other distribution abstractions. The new abstractions should include features of MapReduce that proved to lower the learning curve for new programmers, made distribution of work and data effortless and efficient with cluster topology knowledge, and provided a certain level of fault tolerance. From the work with workflows, the following features would satisfy the requirements of other scientific applications: structures for complex interactions between low-level tasks, scheduling algorithms, operation on multiple data sources, non-textual data formats, and workflow checkpoint. Formulating a set of guidelines for sequential application developers to produce loosely-coupled modular software that fits the low-level abstractions well will be essential to facilitate the adoption of new abstractions.

Extending PQR2

PQR2 improved accuracy of resource usage predictions in scenarios including bioinformatics applications. While various questions have been answered, others for future endeavors remains, including: performance comparison of ML algorithms with respect to

training time and prediction time, optimization in scenarios with large amount of historic information, optimization in scenarios with large amount of nominal attribute values, and collection of monitoring information.

Time for generating PQR model is expected to be the sum of training all classification and regression algorithms multiple times (depending on the number of tree node and leaves as well as number of split locations selected), and prediction time is similar to other non-instance-based algorithms, but dependent on the height of the tree. While this may seem discouraging, training can be performed offline without impact on the prediction time performance, and room for optimization still exist, including parallel implementation, adaptive mechanism to pre-select classification and regression algorithms from the pool, and selection of split location independent of classification algorithm. The offline training update frequency can be maintained by an autonomous system that detects when predictions start to deteriorate.

Large cloud providers are expected to execute hundreds of thousands of tasks a day. Dealing with this amount of potential historic data will be challenging due to the increase in storage and memory requirements when operating on this dataset. Filtering, clustering, and caching [72] are some of the techniques to be applied to reduce the amount of work performed by ML algorithms.

It is also possible that some application or resource attributes will present a large number of nominal values (e.g., when considering user name as a learning attribute). The observation that some algorithms suffered from performance degradation and required a high amount of memory to work with this type of data compels further investigation.

Finally, efforts to provide resource monitoring information on clouds are underway. For example, Amazon CloudWatch offers metrics such as CPU utilization, disk reads and writes, and

network traffic of single instances. However, information about usage of shared resources such as network and storage servers are not readily available. In private clouds where such information can be accessed from network switches and tools such as nfswatch, it is still challenging to effectively collect this information on a per task basis without adding overheads to the system.

APPENDIX A
SCHEDULING ALGORITHMS

Considering the set of jobs (J1 to J6) with features depicted in Figure 4-1 as an example of a set of jobs in the queue, the next sections discuss different scheduling policies and present schedules obtained for each policy. The system being considered consists of three machines, each with two processors and two gigabytes of memory. The scheduler is responsible for solving the bin-packing problem in three dimensions: processing capacity, memory and time. In the comparison graphs, time is shown in the horizontal axis whereas processing capacity and memory are merged as the vertical axis in a manner that the worst characteristic takes precedence. For example, although J3 requires a single processor, its memory requirement (equal to the total amount available in a single node) demands exclusive access to a node, essentially consuming two processors, one of which remains unused. To better discuss the differences among schedulers, the definition of user- and system-centric performance metrics used by previous works are listed first.

Scheduler performance metrics

Makespan: also called time-to-solution, is time between the submission of the first job of a workload and the end of the execution of the last running job. Throughput, the number of jobs executed per unit of time of a workload provides similar information. From the system's perspective the objective is to offer low makespan or high throughput as both metric indicate the overall performance of the system.

Waiting time: is the time spent by the job in the queue waiting for required resources to become available. Low average of this metric indicates better user satisfaction, whereas low maximum waiting time indicates the fairness of the system with respect to the amount of time each jobs waits. When all processing resources are equal, average response time (sum of job's waiting time and execution time) differs from waiting time by a constant (the average runtime). Since execution time is independent of the scheduler, the average runtime remains the same for different non-preemptive schedulers.

Slowdown: is the response time of a job normalized by its execution time on allocated processors. Slowdown indicates how much jobs are delayed compared to their lengths [45][97][144][145]. Since slowdown takes into consideration the execution time irrespective of the amount of processors allocated, it treats differently jobs that require the

same amount of work but using a different number of processors, as noted in [150] and discussed in [45]. Furthermore, since the average of this metric can be highly affected by the existence of very short or failed jobs, several studies [46][94][116][123][132][150] used bounded slowdown metric [44], which increases the execution time of short jobs to a given threshold (typically 10 seconds).

Processor utilization: percentage of processors allocated for scheduled jobs. Indicates how much room for improvement there still exists.

Scheduler job orderings

First Come First Served (FCFS): Jobs in the queue are maintained according to its arrival time and a job at the head of the queue gets scheduled as soon as enough resources become available. Due to its simplicity, it is often used as a reference for comparison with other approaches. While this type of scheduler is easy to implement and offers fairness with respect to waiting time, it usually results in a system with poor performance. This is especially true when the workload contains large jobs that block other smaller jobs, leaving large amount unused resources [2][15][94][123][144][150]. Figure A-1 shows jobs J2 and J4 being responsible for leaving two hosts unused while jobs J1 and J3 are executing.

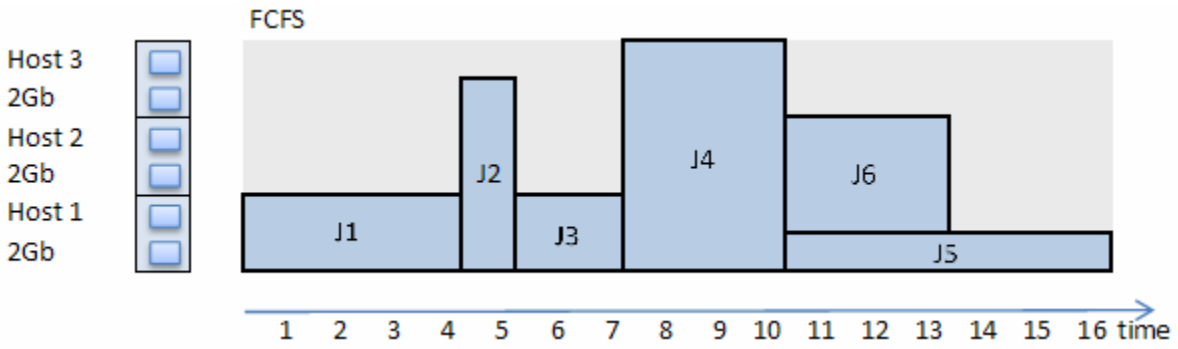


Figure A-1. Schedule obtained from applying First-Come First-Served ordering policy.

Random: Jobs are randomly picked from the queue to be executed next. A random ordering policy serves as a simple mechanism to avoid a burst of jobs from a single user to monopolize the system. The consequence of the randomness is the increased difficulty in predicting the waiting time for this type of system. Simulations in [15] reveal that random

ordering results in increased maximum waiting time than FCFS when only sequential jobs are considered. Figure A-2 shows a random order that was particularly fortunate, resulting in better system utilization than FCFS, but it could equally have resulted in a poorer system, for example if job J5 had been chosen before job J4.

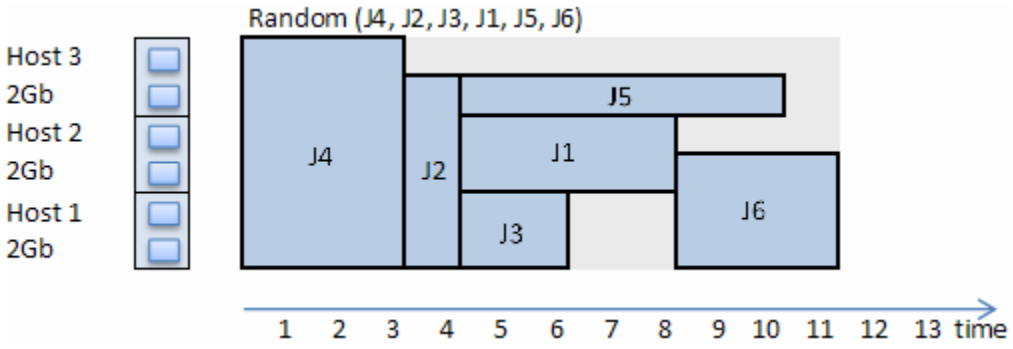


Figure A-2. Schedule obtained from applying random ordering policy.

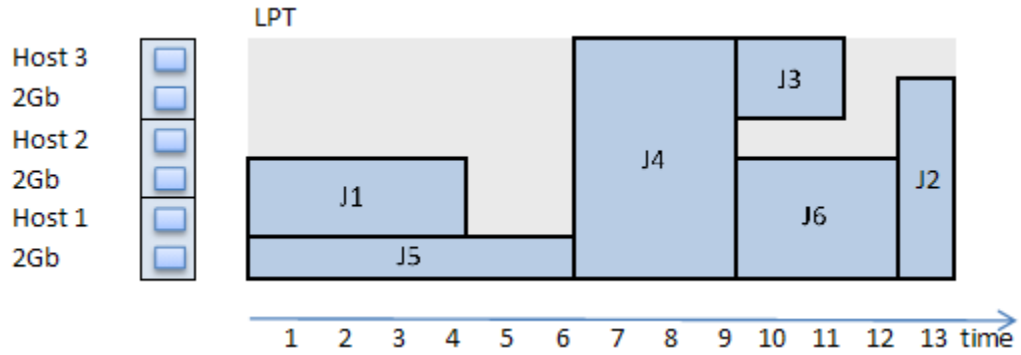


Figure A-3. Schedule obtained from applying LPT ordering policy.

Longest Processing Time First (LPT): Jobs in the queue are ordered according to the requested execution time, from the longest to the shortest, and then by arrival time for jobs with same requested time. The goal of this policy is to reduce the makespan of the system by ensuring that long jobs will not be started close to the end of a workload [15]. Note in Figure A-3 that most of unused resources are left at the beginning of the schedule rather than at the end. Note also that although J3 requires only one processor, J3 cannot be co-scheduled in the same host as J6 since J3 requires all the memory available in a node. The main drawback of this ordering

algorithm is that short jobs can potentially suffer from starvation. For example, in Figure A-3, J2 and J3 could be delayed indefinitely if jobs requiring more than 2 time units kept arriving at steady rate.

Shortest Processing Time First (SPT): As opposed to LPT, jobs requesting shorter execution time are scheduled first. By privileging short jobs, this policy results in a system with better average response time and average slowdown since jobs in the queue do not need to wait for long jobs and short jobs tend to have smaller waiting time [15]. Unfairness against long jobs, is the main drawback of this ordering algorithm [25][26]. In this case, J1 and J5 in Figure A-4 could be delayed indefinitely with the steady arrival of jobs shorter than 4 time units.

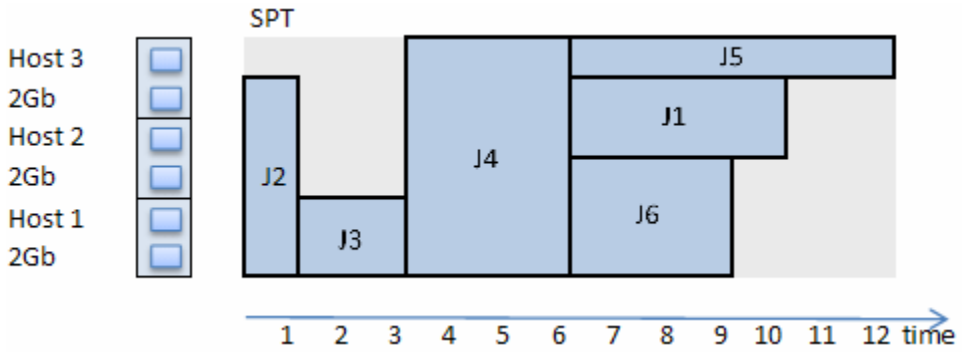


Figure A-4. Schedule obtained from applying SPT ordering policy.

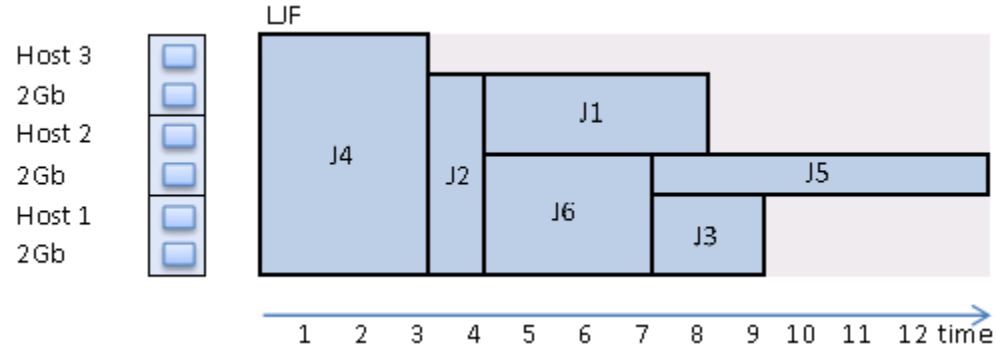


Figure A-5. Schedule obtained from applying LJF ordering policy.

Largest Job First (LJF): jobs in the queue are sorted in nonincreasing order according to the number of processors requested and independent of the execution time required [79]. As

Figure A-5 shows, the idea is that as the size of jobs decreases, more jobs can be fit at the same time without having to compute all permutations of jobs to find the ideal scheduling.

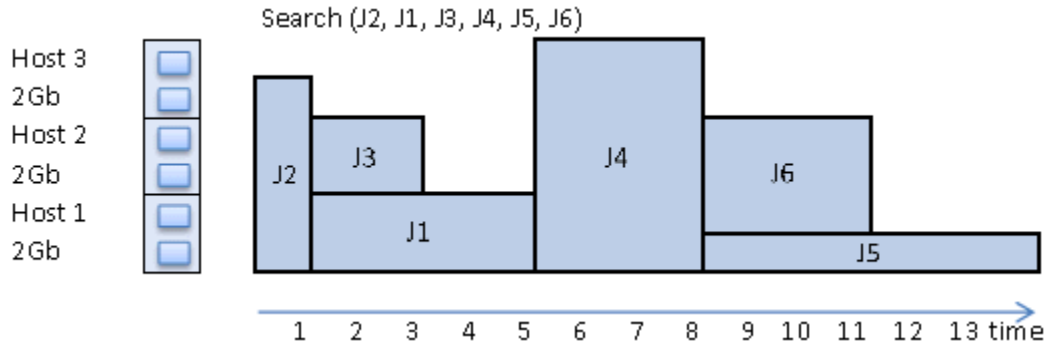


Figure A-6. Schedule obtained searching all permutations with one discrepancy.

Best ordering: schedulers that search the entire space of possible job orderings to find the schedule that provides the best result according to some objective are in this category. The main disadvantage is the exponential computational demand of the search algorithm as the number of jobs to be considered increases. For example, the scenario with just 6 jobs would require the algorithm to consider 720 orderings. This often leads to the creation of heuristics that limit the search space. Two different heuristics are investigated in [27][136]: (a) *limited discrepancy search* considers first the orderings with small discrepancies when compared to a base ordering, and (b) *depth-bound discrepancy search* prioritizes discrepancies in the beginning of the ordering when compared to a base ordering. In [27], the search computation is further improved by pruning cases with a performance worse than the best schedule found so far, and by considering different base ordering dynamically. Depth-bound discrepancy search is shown to provide better performance when a two-level objective is considered. The first level minimizes the total excessive wait time (wait time above some threshold) while the second level minimizes the average slowdown for those schedules with same excessive wait time. Figure A-6 shows the schedule chosen by both search algorithms described in [136], optimizing the average slowdown

(excessive wait threshold higher than any wait time) and limiting the search to the 16 cases (out of 720) that provides a single discrepancy with respect to the initial order by arrival time.

Other complex orderings: while the previous ordering schemes made use of a specific job characteristic, the use of weighted combination of these characteristics (e.g., estimated job execution time, number of processors, arrival time, slowdown, and user priority) to determine the order in which the jobs are considered for execution has also been proposed. Largest eXpansion Factor (LXF) prioritizes jobs with high slowdown [109][123]. LXF&W weights expansion factor and job wait time, and LXF&W&P weights expansion factor, wait time, and number of processors [25]. In addition, [26] proposed and evaluated SPT&W[1] (weights normalized inverse of requested runtime and waiting time), S$\sqrt{\text{PT}}$&W (square root of normalized inverse of requested runtime and waiting time), and L$\sqrt{\text{X}}$F&W (square root of slowdown and waiting time).

Scheduler backfilling

In practice, ordering policies based on a fixed set of job characteristics is preferred since the computational demand of best ordering schedulers can be detrimental to the overall performance of a batch system. However, orderings based on job characteristics lead to schedules with unused resources. Backfilling procedures attempt to fill these unused resources with jobs in the queue, presenting significant improvements in the utilization of space-shared systems. Backfilling policies are characterized by two attributes: the number of reservations accepted by the scheduler and the ordering in which jobs are considered for backfill. Reservations guarantee that jobs close to the head of the queue are not postponed indefinitely. The number of reservations offered by a solution represents a balance between fairness and system utilization and [26] has empirically shown that 2 to 4 reservations offers a good

[1] Original name was changed from SJF&W to SPT&W to match the Shortest Processing Time First (SPT) acronym used in this work.

compromise between these two factors. Backfilling strategies are described next considering a FCFS ordering.

First Fit (no reservation): when there are not enough resources for the first job in the queue, the scheduler searches the queue for jobs that require fewer resources than the amount of resources available, and selects the first job that meets this constraint. In Figure A-7, after scheduling J1, 4 free processors are not sufficient for J2, but they are enough for J3 and J5. When J3 finishes, free resources are enough to run J6. This algorithm presents good system utilization and slowdown performance at the expense of wide jobs such as J4, which would suffer from starvation if the system continued to receive small jobs.

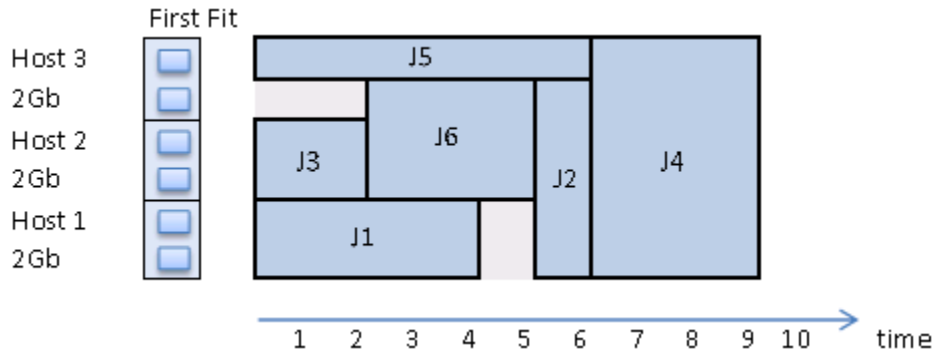


Figure A-7. Schedule obtained from applying First Fit-backfill policy.

Extensible Argonne Scheduling sYstem (EASY): proposed in [80] and implemented in [116], EASY backfilling allows jobs further in the queue to be executed ahead of time as long as the first job in the queue does not get delayed. To guarantee that the first job does not get delayed, EASY offers a single reservation to be placed for the first job when enough resources are not available. In Figure A-8, unlike First Fit backfilling, EASY backfilling policy does not allow J6 to be executed ahead of J2. A reservation for J2 guarantees that J2 is not delayed more than it would in a FCFS policy. Note that since the single reservation is taken by J2, J5 is allowed to delay J4.

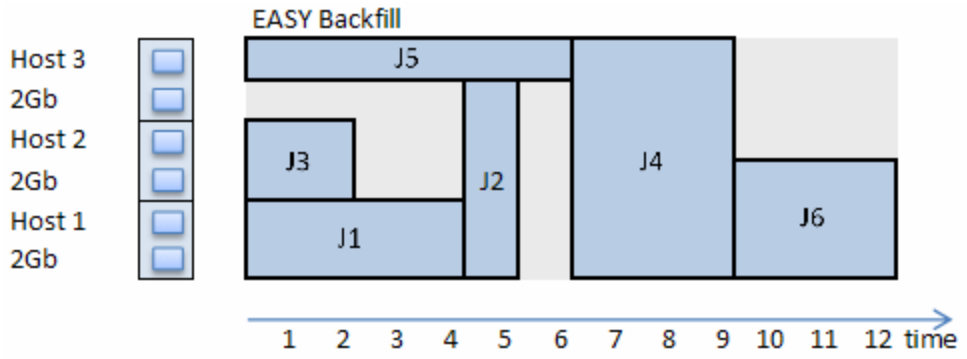


Figure A-8. Schedule obtained from applying EASY-backfill policy.

Conservative: jobs that execute ahead of time cannot delay any of the jobs that are ahead in the queue. In Figure A-9, not only J6 cannot delay J2, but J5 also cannot execute ahead of time delaying J4. In this policy, a reservation is made for every job that is waiting in the queue.

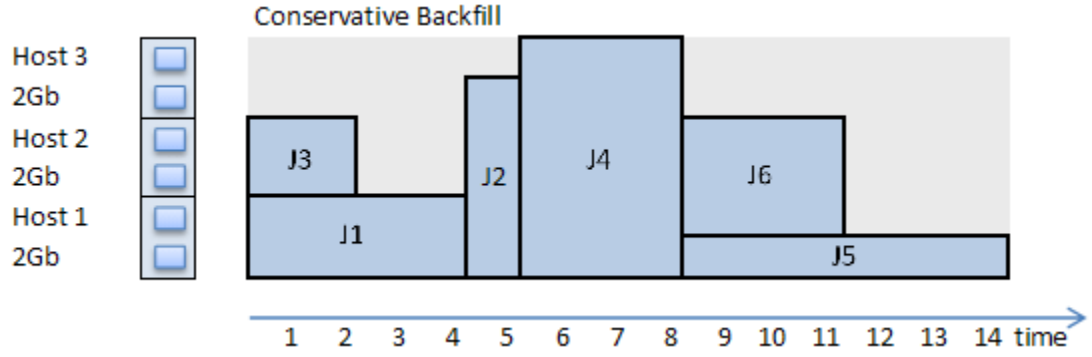


Figure A-9. Schedule obtained from applying conservative-backfill policy.

The backfill ordering can be any of the orderings described earlier and may be different from the main scheduling ordering policy. For example, a FCFS/SPT-backfill scheduler orders job according to arrival time, but backfills shortest jobs first, with the goal of decreasing the average slowdown metric. Another example shown below combines FCFS ordering with a best ordering backfilling scheme.

Lookahead Optimizing Scheduler (LOS): proposed in [115], LOS is a single reservation backfill policy that chooses the best combination of jobs to be backfilled in a manner that system utilization is maximized. The best combination is chosen using a dynamic programming

approach. As shown in Figure A-10**,** LOS chooses J6 instead of J3 to run ahead of J2 as J6 maximizes the system utilization when compared to J3. The drawback of this algorithm is the computation demanded to traverse the combination of jobs when the size of the queue grows. Heuristics to limit the number of jobs to consider can be applied to reduce the computational demand.

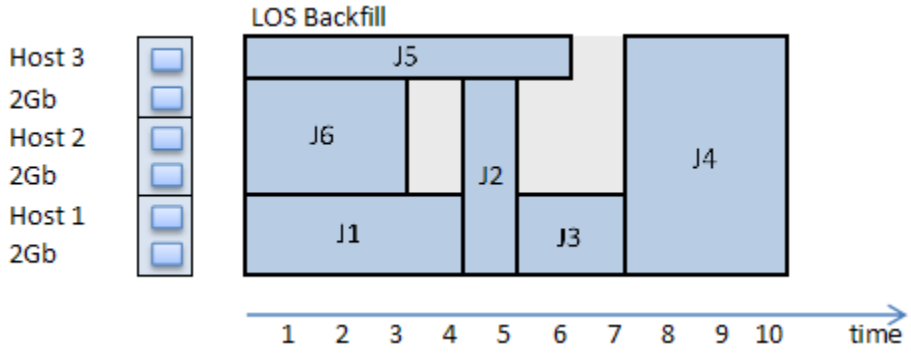


Figure A-10. Schedule obtained from applying LOS-backfill policy.

Scheduler with job preemption

Preemption policy determines which jobs should be preempted to favor another job with higher priority. In practice, creating an environment with job preemption and migration from one host to another requires additional programming and incurs performance overhead. Clouds with VM live migration solutions (e.g., Citrix XenMotion, Microsoft System Center Virtual Machine Manager, and VMware VMotion) eliminate the need to modify applications. Nonetheless, suspending, migrating and resuming a VM also consume resources. Schedulers armed with knowledge about applications resource usage and preemption costs are in a better position to make better automated preemption decisions.

Schedulers in Distributed Computing Environments

When considering distributed systems, the single queue model depicted in Figure 4-1 needs to be expanded. Existing solutions in grid systems can be classified taking into

consideration the location of the scheduler (local with global dispatcher or global), the location of the re-scheduler (local or global), and the direction of the communication (push or pull). Figure A-11 illustrates the differences between a global and a local scheduler. The global scheduler design [108] is just an extension of a local centralized scheduler applied to various sites, usually yielding good performance as techniques such as backfilling can be easily applied. In this model, users submit their jobs to the global scheduler and the distinct sites are only responsible for running the jobs, effectively losing control over the policy of job execution. Often, sites give preference to the local scheduler model [60][146] since each site can maintain its own scheduling policies. For users with access to multiple sites, a global dispatcher can observe the status of each site and decide based on best predicted response time, to which site the job should be submitted. There are also variations of dispatcher that submit the same job to multiple possible sites and once the dispatcher is informed that the job started execution in one of these sites, the job on the other sites are cancelled [127]. The impact on the performance of schedulers and middleware due to the increase in the number of jobs to be processed was evaluated in [23].

When scheduling decisions are local, sites need to allow some redistribution of jobs in order for the collection of sites to take advantage of unused resources. This re-scheduling can be performed locally or globally. In the global model, a centralized re-scheduler collects queue information and current allocation from all sites and verifies the possibility of migrating queued jobs from one site to another [146]. In the local model, the exchange is performed between the sites in a peer-to-peer fashion, without a central moderator [59]. While a global re-scheduler presents the potential of finding better mix of jobs due to its global knowledge, the local re-scheduler provides flexibility in defining policies by the collaborating sites.

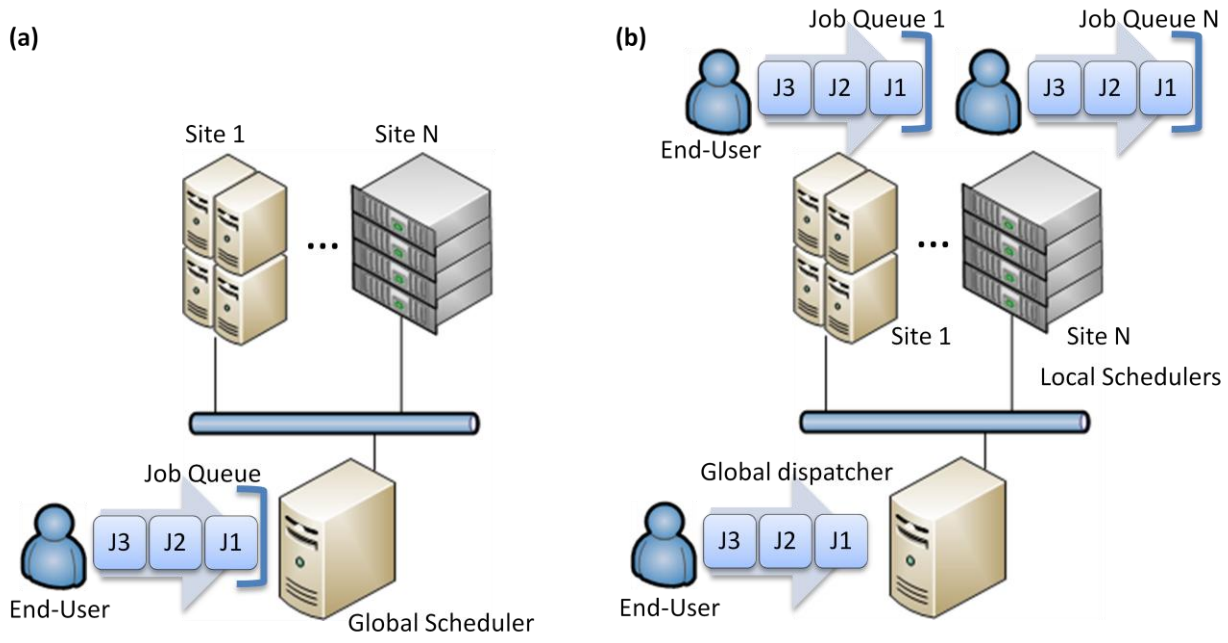


Figure A-11. Global and local schedulers. Global scheduler (a) is responsible for receiving all jobs and maintaining the information about current job allocations. Local schedulers (b) work independently receiving jobs only from local users. For users with accounts in multiple sites, a global dispatcher can choose the best local queue and submit the job to it.

The communication between sites and between global and local systems can be performed in two directions. A global scheduler can push a job to a site or wait for the site to ask for a job. A local re-scheduler can offer queued jobs to be migrated or can advertise the possibility of receiving a job.

Grid systems utilizing space-shared schedulers as just presented also benefit from the job's resource demand prediction since schedulers continue to depend on this information to decide when and where each job should be executed in an efficient manner. In addition, the predicted execution time of the jobs in the queue can, in turn, be used to estimate the expected wait time of the job in the queue. The scheduler or re-scheduler can then decide to which site a job should be assigned or if job migration can yield better result. A per-user global scheduler proposed in [61] uses predicted execution time to estimate the job wait time in each site before submitting the job

to a site. The average wait time and slowdown of the system was shown to improve as the number of sites that can be considered for job execution increases. In time-shared grid systems multiple jobs can be executed simultaneously on the same resource without the need of a queue. In this scenario, the prediction of resource consumption is not as important as the prediction of resource's load over time and its effect for the executing jobs. A noteworthy example of resource load forecaster is the network weather service (NWS) [143], which chooses the best model within several (e.g., mean, median, and autoregressive) to predict CPU, memory, and network utilization from past data viewed as a time series. Exponential smoothing technique was applied in [35] to predict processing speed for the next iteration of a parallel application, achieving on average a 1.8 speedup as work got rebalanced according to the new predictions.

LIST OF REFERENCES

[1] 3tera, Inc. 3tera. [Online]. Available: http://www.3tera.com. Accessed: March/2010.

[2] K. Aida, “Effect of Job Size Characteristics on Job Scheduling Performance,” in Proc. Workshop Job Scheduling Strategies Parallel Process., 2000, pp. 1-17.

[3] R. Albers, E. Suijs, and P. H. N. de With, “Triple-C: Resource-usage prediction for semi-automatic parallelization of groups of dynamic image-processing tasks,” in Proc. 23rd Int. Parallel Distributed Processing Symp., 2009.

[4] G. Allen, K. Davis, T. Goodale *et al.*, “The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for the Grid,” *Proc. IEEE,* vol. 93, no. 3, pp. 534-550, 2005.

[5] E. Alpaydin, *Introduction to machine learning*, MIT Press, 2004.

[6] S. F. Altschul, W. Gish, W. Miller *et al.*, “Basic Local Alignment Search Tool,” *J. Molecular Biology*, vol. 215, no. 3, pp. 403-410, 1990.

[7] Amazon Web Services LLC. Amazon Elastic Compute Cloud. [Online]. Available: http://aws.amazon.com/ec2. Accessed: March/2010.

[8] Amazon Web Services LLC. Amazon Simple Storage Service. [Online]. Available: http://aws.amazon.com/s3. Accessed: March/2010.

[9] J. Andrade, M. Andersen, L. Berglund *et al.*, "Applications of Grid Computing in Genetics and Proteomics," *Applied Parallel Computing. State of the Art in Scientific Computing*, pp. 791-798, 2008.

[10] Apache Software Foundation. Apache Axis2. [Online]. Available: http://ws.apache.org/axis2. Accessed: March/2010.

[11] Apache Software Foundation. Apache Hadoop project. [Online]. Available: http://hadoop.apache.org. Accessed: March/2010.

[12] Apache Software Foundation. Apache HTTP Server Project. [Online]. Available: http://httpd.apache.org. Accessed: March/2010.

[13] Apache Software Foundation. Apache Tomcat. [Online]. Available: http://tomcat.apache.org. Accessed: March/2010.

[14] M. Armbrust, A. Fox, R. Griffith *et al.*, “Above the Clouds: A Berkeley View of Cloud Computing,” EECS Department, University of California, Berkeley, UCB/EECS-2009-28, 2009.

[15] O. Arndt, B. Freisleben, T. Kielmann *et al.*, "A comparative study of online scheduling algorithms for networks of workstations," *Cluster Computing*, vol. 3, no. 2, pp. 95-112, 2000.

[16] P. Balaji, W. Feng, H. Lin *et al.*, "Distributed Data I/O with ParaMEDIC: Experiences with a Worldwide Supercomputer," in Proc. Int. Supercomputing Conf., Dresden, Germany, 2008.

[17] K. Ballinger, D. Ehnebuske, C. Ferris *et al.*, "Basic Profile Version 1.1," WS-I Organization, 2004.

[18] P. Barham, B. Dragovic, K. Fraser *et al.*, "Xen and the art of virtualization," in Proc. 19th ACM Symp. Operating Systems Principles, Bolton Landing, NY, USA, 2003, pp. 164-177.

[19] C. Baru, R. Moore, A. Rajasekar *et al.*, "The SDSC storage resource broker," in Proc. 1998 Conf. Centre for Advanced Studies Collaborative research, Toronto, Ontario, Canada, 1998, p. 5.

[20] J. Bi, and K. P. Bennek, "Regression error characteristic curves," in Proc. 20th Int. Conf. Machine Learning, Washington DC, 2003, pp. 43-50.

[21] R. Bolze, F. Cappello, E. Caron *et al.*, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *Int. J. High Performance Computing Applications*, vol. 20, no. 4, pp. 481-494, November, 2006.

[22] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in Proc. fifth annual workshop on Computational learning theory, Pittsburgh, Pennsylvania, United States, 1992, pp. 144-152.

[23] H. Casanova, "On the Harmfulness of Redundant Batch Requests," in Proc. 15th IEEE Int. Symp. High Performance Distributed Computing, 2006, pp. 255-266.

[24] V. Cavalli-Sforza, J. G. Carbonell, and P. J. J. Jansen, "Developing Language Resources for Transnational Digital Government Systems: A Case Study," in Proc. 4th Int. Conf. Language Resources Evaluation, Lisbon, Portugal, 2004, pp. 945-948.

[25] S.-H. Chiang, and M. K. Vernon, "Production Job Scheduling for Parallel Shared Memory Systems," in Proc. 15th Int. Parallel Distributed Processing Symp., 2001, p. 47.

[26] S.-H. Chiang, A. C. Arpaci-Dusseau, and M. K. Vernon, "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance," in Proc. Workshop Job Scheduling Strategies Parallel Process., 2002, pp. 103-127.

[27] S.-H. Chiang, and S. Vasupongayya, "Design and Potential Performance of Goal-Oriented Job Scheduling Policies for Parallel Computer Workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 12, pp. 1642-1656, 2008.

[28] B. Cohen, "Incentives Build Robustness in BitTorrent," in Proc. 1st Workshop Economics Peer-to-Peer Systems, 2003.

[29] Condor team. Condor Project. [Online]. Available: http://www.cs.wisc.edu/condor. Accessed: March/2010.

[30] Cycle Computing, LLC. Cycle Computing CycleCloud. [Online]. Available: http://www.cyclecomputing.com. Accessed: March/2010.

[31] A. E. Darling, L. Carey, and W.-C. Feng, "The Design, Implementation, and Evaluation of mpiBLAST," in Proc. 4th Int. Conf. Linux Cluster, 2003.

[32] J. Dean, and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in Proc. 6th Conf. Symp. Opearting Systems Design Implementation, San Francisco, CA, 2004, pp. 137-150.

[33] A. Denis, O. Aumage, R. Hofman *et al.*, "Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems," in Proc. 13th IEEE Int. Symp. High Performance Distributed Computing, 2004, pp. 97-106.

[34] M. D. Dikaiakos, D. Katsaros, P. Mehra *et al.*, "Cloud Computing: Distributed Internet Computing for IT and Scientific Research," *IEEE Internet Computing*, vol. 13, no. 5, pp. 10-13, September/October, 2009.

[35] M. Dobber, G. Koole, and R. v. d. Mei, "Dynamic load balancing experiments in a grid," in Proc. 5th IEEE Int. Symp. Cluster Computing Grid, 2005, pp. 1063-1070.

[36] H. Drucker, C. J. C. Burges, L. Kaufman *et al.*, "Support Vector Regression Machines," in Advances in Neural Information Processing Systems 9, 1997, pp. 155—161.

[37] R. Duan, F. Nadeem, J. Wang *et al.*, "A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids," in Proc. 2009 9th IEEE/ACM Int. Symp. Cluster Computing and the Grid, 2009, pp. 339-347.

[38] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2 ed., Wiley, 2001.

[39] R. Durbin, S. R. Eddy, A. Krogh *et al.*, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*, Cambridge University Press, 1998.

[40] R. A. V. Engelen, and K. A. Gallivan, "The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks," in Proc. 2nd IEEE/ACM Int. Symp. Cluster Computing Grid, 2002.

[41] Engine Yard, Inc. Engine Yard. [Online]. Available: http://www.engineyard.com. Accessed: March/2010.

[42] European Bioinformatics Institute. UniProt/TrEMBL database. [Online]. Available: http://www.ebi.ac.uk/uniprot/TrEMBLstats. Accessed: March/2010.

[43] D. Feitelson. Parallel Workloads Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload. Accessed: March/2010.

[44] D. G. Feitelson, and L. Rudolph, “Metrics and Benchmarking for Parallel Job Scheduling,” in Proc. Workshop Job Scheduling Strategies Parallel Process., 1998, pp. 1-24.

[45] D. G. Feitelson, “Metrics for Parallel Job Scheduling and Their Convergence,” in Proc. Workshop Job Scheduling Strategies Parallel Process., 2001, pp. 188-206.

[46] D. G. Feitelson, “Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 175-182, 2005.

[47] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, “A case for grid computing on virtual machines,” in Proc. 23rd Int. Conf. Distributed Computing Systems, 2003, pp. 550-559.

[48] I. Foster, and C. Kesselman, "The Grid 2: Blueprint for a New Computing Infrastructure," Morgan Kaufmann Publishers Inc., 2003.

[49] A. Ganguly, A. Agrawal, P. Boykin *et al.*, “WOW: Self-organizing Wide Area Overlay Networks of Virtual Workstations,” *J. Grid Computing*, vol. 5, no. 2, pp. 151-172, 2007.

[50] R. Gibbons, “A Historical Application Profiler for Use by Parallel Schedulers,” in Proc. Workshop Job Scheduling Strategies Parallel Process., 1997, pp. 58-77.

[51] B. Gleeson, A. Lin, J. Heinanen *et al.*, “A Framework for IP Based Virtual Private Networks,” RFC2764, RFC Editor, 2000.

[52] Globus Alliance. Globus Toolkit. [Online]. Available: http://www.globus.org/toolkit. Accessed: March/2010.

[53] Globus Alliance. Nimbus toolkit. [Online]. Available: http://workspace.globus.org/. Accessed: March/2010.

[54] Globus Alliance. Science Clouds. [Online]. Available: http://workspace.globus.org/clouds. Accessed: March/2010.

[55] Globus Security Team, “Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective,” Globus Alliance, 2005.

[56] L. Gong, “Project JXTA: A Technology Overview,” Sun Microsystems Inc., 2002.

[57] Google, Inc. Google App Engine. [Online]. Available: http://code.google.com/appengine. Accessed: March/2010.

[58] Google, Inc. Google Apps. [Online]. Available: http://www.google.com/apps. Accessed: March/2010.

[59] C. Grimme, J. Lepping, and A. Papaspyrou, "Discovering performance bounds for grid scheduling by using evolutionary multiobjective optimization," in Proc. 10th Annu. Conf. Genetic evolutionary computation, Atlanta, GA, USA, 2008, pp. 1491-1498.

[60] F. Guim, and J. Corbalan, "A Job Self-scheduling Policy for HPC Infrastructures," in Proc. Workshop Job Scheduling Strategies Parallel Process., 2008, pp. 51-75.

[61] F. Guim, I. Rodero, M. Garcia *et al.*, "The XtreemOS jScheduler: using self-scheduling techniques in large computing architectures," in Proc. 1st USENIX Workshop Large-Scale Computing, Boston, MA, 2008, pp. 1-10.

[62] C. Gupta, A. Mehta, and U. Dayal, "PQR: Predicting Query Execution Times for Autonomous Workload Management," in Proc. 2008 Int. Conf. Autonomic Computing, 2008, pp. 13-22.

[63] IBM. WebSphere Enterprise Service Bus. [Online]. Available: http://www-306.ibm.com/software/integration/wsesb. Accessed: March/2010.

[64] Intel, "Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors," Intel, 2008.

[65] M. Isard, M. Budiu, Y. Yu *et al.*, "Dryad: distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59-72, 2007

[66] M. Janssen, and A. Cresswell, "Enterprise Architecture Integration in E-Government," in Proc. 38th Annu. Hawaii Int. Conf. System Sciences, 2005.

[67] G. H. John, and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in Proc. 11th Conf. Uncertainty in Artificial Intelligence, 1995, pp. 338-345.

[68] M. Tim Jones. Cloud computing with Linux: Cloud computing platforms and applications. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-cloud-computing. Accessed: March/2010.

[69] D. Joseph, J. Kannan, A. Kubota *et al.*, "OCALA: An Architecture for Supporting Legacy Applications over Overlays," in Proc. 3rd Symp. Networked Systems Design Implementation, 2006, pp. 267–280.

[70] Joyent, Inc. Joyent. [Online]. Available: http://joyent.com. Accessed: March/2010.

[71] G. Kandaswamy, L. Fang, Y. Huang *et al.*, "Building web services for scientific grid applications," *IBM Journal of Research and Development*, vol. 50, no. 2/3, pp. 249-260, 2006.

[72] N. H. Kapadia, "On the Design of a Demand-Based Network-Computing System: The Purdue University Network-Computing Hubs," Purdue University, 1999.

[73] K. Keahey, I. Foster, T. Freeman *et al.*, "Virtual workspaces: Achieving quality of service and quality of life in the Grid," *Sci. Program.,* vol. 13, no. 4, pp. 265-275, 2005.

[74] K. Keahey, and T. Freeman, "Contextualization: Providing One-Click Virtual Clusters," in Proc. Fourth IEEE Int. Conf. eScience, 2008, pp. 301-308.

[75] H. Kreger, "Web Service Conceptual Architecture (WSCA 1.0)," IBM, 2001.

[76] A. Krishnan, "GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework: Research Articles," *Concurr. Comput.: Pract. Exper.,* vol. 17, no. 13, pp. 1607-1623, 2005.

[77] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky, "Estimating computation times of data-intensive applications," *IEEE Distributed Systems Online,* vol. 5, no. 4, 2004.

[78] KVM. Kernel-based Virtual Machine. [Online]. Available: http://www.linux-kvm.org. Accessed: March/2010.

[79] K. Li, and K. H. Cheng, "Job Scheduling in a Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme," *IEEE Trans. Parallel Distrib. Syst.,* vol. 2, no. 4, pp. 413-422, 1991.

[80] D. Lifka, "The ANL/IBM SP scheduling system," in Proc. Workshop Job Scheduling Strategies Parallel Process., 1995, pp. 295-303.

[81] B. Ludäscher, I. Altintas, C. Berkley *et al.*, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience,* vol. 18, no. 10, pp. 1039-1065, 2006.

[82] J. Maassen, and H. E. Bal, "Smartsockets: solving the connectivity problems in grid computing," in Proc. 16th IEEE Int. Symp. High Performance Distributed Computing, Monterey, California, USA, 2007, pp. 1-10.

[83] M. Matsuda, T. Kudoh, Y. Kodama *et al.*, "Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks," in Proc. 2006 IEEE Int. Conf. Cluster Computing, 2006, pp. 1-9.

[84] A. Matsunaga, M. Tsugawa, S. Adabala *et al.*, "Science gateways made easy: the In-VIGO approach: Research Articles," *Concurr. Comput.: Pract. Exper.,* vol. 19, no. 6, pp. 905-919, 2007.

[85] A. Matsunaga. BLAST and RAxML execution traces. [Online]. Available: http://www.acis.ufl.edu/prediction. Accessed: March/2010.

[86] A. Mehta, C. Gupta, S. Wang *et al.*, "Automatic Workload Management for Enterprise Data Warehouses," *Bulletin IEEE Comput. Soc. Tech. Committee on Data Eng.,* vol. 31, no. 1, pp. 11-19, 2008.

[87] Microsoft Corporation. Microsoft Online Services. [Online]. Available: http://www.microsoft.com/online. Accessed: March/2010.

[88] Microsoft Corporation. Microsoft Visual Studio. [Online]. Available: http://www.microsoft.com/visualstudio. Accessed: March/2010.

[89] Microsoft Corporation. Microsoft Windows Azure Platform. [Online]. Available: http://www.microsoft.com/windowsazure. Accessed: March/2010.

[90] Microsoft Corporation. Windows Communication Foundation. [Online]. Available: http://msdn.microsoft.com/en-us/netframework/aa663324.aspx. Accessed: March/2010.

[91] Microsoft Corporation. Microsoft Internet Information Services (IIS). [Online]. Available: http://www.iis.net. Accessed: March/2010.

[92] Microsoft Corp., "Windows Server 2008 Hyper-V Technical Overview," 2008.

[93] A. W. Moore, J. Schneider, and K. Deng, "Efficient Locally Weighted Polynomial Regression Predictions," in Proc. 14th Int. Conf. Machine Learning, 1997, pp. 236-244.

[94] A. W. Mu'alem, and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529-543, 2001.

[95] myGrid team. myGrid Website. [Online]. Available: http://www.mygrid.org.uk. Accessed: March/2010.

[96] F. Nadeem, and T. Fahringer, "Using Templates to Predict Execution Time of Scientific Workflow Applications in the Grid," in Proc. 2009 9th IEEE/ACM Int. Symp. Cluster Computing and the Grid, 2009, pp. 316-323.

[97] V. K. Naik, M. S. Squillante, and S. K. Setia, "Performance analysis of job scheduling policies in parallel supercomputing environments," in Proc. 1993 ACM/IEEE Conf. Supercomputing, Portland, Oregon, United States, 1993, pp. 824-833.

[98] Netbeans Community. Netbeans IDE. [Online]. Available: http://www.netbeans.org. Accessed: March/2010.

[99] D. Nurmi, R. Wolski, C. Grzegorczyk *et al.*, "The Eucalyptus Open-source Cloud-computing System," in Proc. 1st Workshop Cloud Computing Its Applications, 2008.

[100] OASIS, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," OASIS Standard Specification, 2006.

[101] OMII-UK. Open Middleware Infrastructure Institute. [Online]. Available: http://www.omii.ac.uk. Accessed: March/2010.

[102] OpenNebula Project. OpenNebula: The Open Source Toolkit for Cloud Computing. [Online]. Available: http://www.opennebula.org. Accessed: March/2010.

[103] Oracle. Oracle WebLogic. [Online]. Available: http://www.oracle.com/appserver. Accessed: March/2010.

[104] D. Passmore, "The Virtual LAN Technology Report," Decisys, Inc., 1996.

[105] PHP group. PHP SOAP extension. [Online]. Available: http://us.php.net/soap. Accessed: March/2010.

[106] RedHat Middleware LLC. JBoss Enterprise Web Server. [Online]. Available: http://www.jboss.com/products/platforms/webserver. Accessed: March/2010.

[107] B. Rochwerger, D. Breitgand, E. Levy *et al.*, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 3, no. 4, pp. 11, 2009.

[108] I. Rodero, F. Guim, J. Corbalan *et al.*, "The Grid Backfilling: a Multi-Site Scheduling Architecture with Data Mining Prediction Techniques," *Grid Middleware and Services*, pp. 137-152, 2008.

[109] G. Sabin, and P. Sadayappan, "Unfairness Metrics for Space-Sharing Parallel Job Schedulers," in Proc. Workshop Job Scheduling Strategies Parallel Process., 2005, pp. 238-256.

[110] Salesforce.com, Inc. Force.com Platform. [Online]. Available: http://www.salesforce.com/platform. Accessed: March/2010.

[111] Salesforce.com, Inc. Web-based Customer Relationship Management (CRM) Software-as-a-Service (SaaS). [Online]. Available: http://www.salesforce.com. Accessed: March/2010.

[112] V. Sanjeepan, A. Matsunaga, L. Zhu *et al.*, "A Service-Oriented, Scalable Approach to Grid-Enabling of Legacy Scientific Applications," in Proc. IEEE Int. Conf. Web Services, 2005, pp. 553-560.

[113] M. Senger, P. Rice, and T. Oinn, "Soaplab - a unified Sesame door to analysis tools," in Proc. UK e-Science All Hands Meeting, 2003, pp. 509-513.

[114] ServePath LLC. GoGrid Cloud Hosting. [Online]. Available: http://www.gogrid.com. Accessed: March/2010.

[115] E. Shmueli, and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1090-1107, 2005.

[116] J. Skovira, W. Chan, H. Zhou *et al.*, "The EASY - LoadLeveler API project," in Proc. Workshop Job Scheduling Strategies Parallel Process., 1996, pp. 41-47.

[117] J. Smith, and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*, Morgan Kaufmann, 2005.

[118] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," *J. Parallel Distrib. Comput.*, vol. 64, no. 9, pp. 1007-1016, 2004.

[119] W. Smith, "Prediction Services for Distributed Computing," in Proc. 21st Int. Parallel Distributed Processing Symp., 2007.

[120] SOAP::Lite development team. SOAP::Lite for Perl. [Online]. Available: http://www.soaplite.com. Accessed: March/2010.

[121] S. Son, and M. Livny, "Recovering Internet Symmetry in Distributed Computing," in Proc. 3rd Int. Symp. Cluster Computing Grid, 2003, pp. 542- 549.

[122] B. Sotomayor, R. S. Montero, I. M. Llorente *et al.*, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14-22, September/October, 2009.

[123] S. Srinivasan, R. Kettimuthu, V. Subramani *et al.*, "Characterization of Backfilling Strategies for Parallel Job Scheduling," in Proc. 2002 Int. Conf. Parallel Processing Workshops, 2002, pp. 514-519.

[124] A. Stamatakis, "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688-2690, Nov 1, 2006.

[125] H. Stockinger, M. Pagni, L. Cerutti *et al.*, "Grid Approach to Embarrassingly Parallel CPU-Intensive Bioinformatics Problems," in Proc. 2nd IEEE Int. Conf. e-Science Grid Computing, 2006, p. 58.

[126] S. Su, J. Fortes, T. R. Kasad *et al.*, "Transnational Information Sharing, Event Notification, Rule Enforcement and Process Coordination," *Int. J. Electronic Government Research*, vol. 1, no. 2, pp. 1-26, 2005.

[127] V. Subramani, R. Kettimuthu, S. Srinivasan *et al.*, "Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests," in Proc. 11th IEEE Int. Symp. High Performance Distributed Computing, 2002, p. 359.

[128] Sun Microsystems. Java Web Services Developer Pack 2.0. [Online]. Available: http://java.sun.com/webservices/docs/2.0. Accessed: March/2010.

[129] A. I. Sundararaj, and P. A. Dinda, "Towards virtual networks for virtual machine grid computing," in Proc. 3rd Conf. Virtual Machine Research Technology Symp., San Jose, California, 2004, pp. 177–190.

[130] M. Tatezono, N. Maruyama, and S. Matsuoka, "Making Wide-Area, Multi-site MPI Feasible Using Xen VM," *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, pp. 387-396, 2006.

[131] O. Thorsen, B. Smith, C. P. Sosa *et al.*, "Parallel Genomic Sequence-Search on a Massively Parallel System," in Proc. 4th Int. Conf. Computing Frontiers, Ischia, Italy, 2007, pp. 59-68.

[132] D. Tsafrir, "Modeling, Evaluating, and Improving the Performance of Supercomputer Scheduling," Computer Science and Engineering, Hebrew University, 2006.

[133] M. Tsugawa, and J. A. B. Fortes, "A virtual network (ViNe) architecture for grid computing," in Proceedings of the 20th International Parallel and Distributed Processing Symposium, 2006, p. 10.

[134] University of Florida. ACIS cloud. [Online]. Available: http://www.acis.ufl.edu/vws. Accessed: March/2010.

[135] L. M. Vaquero, L. Rodero-Merino, J. Caceres *et al.*, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50-55, 2009.

[136] S. Vasupongayya, C. Su-Hui, and B. Massey, "Search-based Job Scheduling for Parallel Computer Workloads," in IEEE Int. Cluster Computing, 2005, pp. 1-10.

[137] VMware Inc., "Virtualization Overview," VMware, 2006.

[138] VMware Inc., "VMware vSphere 4 Competitive Reviewer's Guide," VMware, 2009.

[139] E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing,"*;login: The USENIX Magazine*, vol. 33, no. 5, 2008.

[140] W. Ward, "Understanding spontaneous speech: the Phoenix system," in Proceedings of the 1991 International Conference Acoustics, Speech, and Signal Processing, 1991, pp. 365-367.

[141] I. H. Witten, and E. Frank, *Data mining: practical machine learning tools and techniques*, 2 ed., Morgan Kaufmann, 2005.

[142] R. Wolski, "Dynamically forecasting network performance using the Network Weather Service," *Cluster Computing*, vol. 1, no. 1, pp. 119-132, 1998.

[143] R. Wolski, "Experiences with predicting resource performance on-line in computational grid settings," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 41-49, 2003.

[144] A. K. L. Wong, and A. M. Goscinski, "Evaluating the EASY-backfill job scheduling of static workloads on clusters," in Proc. 2007 IEEE Int. Conf. Cluster Computing, 2007, pp. 64-73.

[145] A. K. L. Wong, and A. M. Goscinski, "The Impact of Under-Estimated Length of Jobs on EASY-Backfill Scheduling," in Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2008, pp. 343-350.

[146] J. Yue, "Global Backfilling Scheduling in Multiclusters," in Proc. 2nd Asian Applied Computing Conf., Kathmandu, Nepal, 2004, pp. 232-239.

[147] Z. Zhang, S. Schwartz, L. Wagner *et al.*, "A greedy algorithm for aligning DNA sequences," *J. Computational Biology,* vol. 7, no. 1/2, pp. 203-214, 2000.

[148] M. Zhao, V. Chadha, and R. J. Figueiredo, "Supporting application-tailored grid file system sessions with WSRF-based services," in Proc. 14th IEEE Int. Symp. High Performance Distributed Computing, 2005, pp. 24-33.

[149] L. Zhu, A. Matsunaga, V. Sanjeepan *et al.*, "Application modeling and representation for automatic grid-enabling of legacy applications," in Proc. First Int. Conf. e-Science Grid Computing, 2005, pp. 24-31.

[150] D. Zotkin, and P. J. Keleher, "Job-Length Estimation and Performance in Backfilling Schedulers," in Proc. 8th IEEE Int. Symp. High Performance Distributed Computing, 1999, pp. 236-243.

BIOGRAPHICAL SKETCH

Andréa Matsunaga received B.S. and M.S. degrees in electrical engineering from the Polytechnic School of the University of São Paulo (Brazil). She engaged in research early, during her undergraduate studies, developing software in the areas of relational databases, control systems and parallel applications. As a master student, she evaluated ATM network interconnects on a Linux cluster with software-based distributed shared memory.

At the Department of Electrical and Computer Engineering of the University of Florida, she has been a research assistant under Dr. Fortes' guidance. At the Advanced Computing and Information Systems (ACIS) Laboratory, she was involved in several projects: ACIS/AIMS information system and web portal, In-Virtual Information Grid Organizations (In-VIGO) Grid middleware, Transnational Digital Government border control system, In-VIGO/nanoHUB collaboration, In-VIGO/ICBR/UF-HPC collaboration, ACIS/Whitney Laboratories collaboration, Atomic-scale Friction Research and Education Synergy Hub and CloudBLAST.