Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds

Eddy Caron, Luis Rodero-Merino, Frédéric Desprez, Adrian Muresan

Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds

Eddy Caron, Luis Rodero-Merino, Frédéric Desprez, Adrian Muresan

UMR CNRS - ENS de Lyon - INRIA - UCB Lyon 5668,
46 allée d'Italie, F-69364 Lyon, France

Universidad Politécnica de Madrid, Facultad de Informática,
B2 L3201, 28660 Boadilla del Monte, Spain

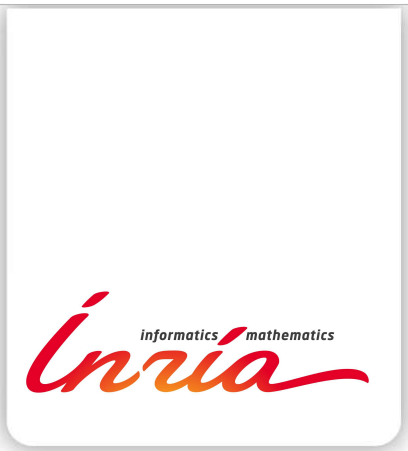
RESEARCH REPORT

N° 7857

January 2012

Project-Team GRAAL

ISSN 0249-6399 ISRN INRIA/RR--7857--FR+ENG



Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds

Eddy Caron*, Luis Rodero-Merino†, Frédéric Desprez‡, Adrian Muresan§

UMR CNRS - ENS de Lyon - INRIA - UCB Lyon 5668,
46 allée d'Italie, F-69364 Lyon, France

Universidad Politécnica de Madrid, Facultad de Informática,
B2 L3201, 28660 Boadilla del Monte, Spain

Project-Team GRAAL

Research Report n° 7857 — January 2012 — 24 pages

* Eddy.Caron@ens-lyon.fr
† lrodero@fi.upm.es
‡ Frederic.Desprez@ens-lyon.fr
§ Adrian.Muresan@ens-lyon.fr

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Abstract: Over the past years, the Cloud phenomenon had an impressive increase in popularity in both the software industry and research worlds. The most interesting feature that Cloud Computing brings, from a Cloud client's point of view, is the on-demand resource provisioning model. This allows Cloud client platforms to be scaled up in order to accommodate more incoming clients and to scale down when the platform has unused resources, and this can all be done while the platform is running. As a result, the physical resources are used more efficiently and the Cloud client saves expenses.
Achieving the above mentioned is not trivial and is done by leveraging more direct functionalities that Clouds provide. Three of these key functionalities are automatic scaling, load balancing and monitoring. They represent the focus of the current work.
This report is divided into three main parts, one for each of the three main topics of interest: auto-scaling, load balancing and monitoring. We detail each of the three topics and present details on their presence and implementation in the commercial, open-source and research worlds.
Among the commercial Cloud platform providers we have focused on Amazon EC2, Microsoft Azure, GoGrid and RackSpace. From the open-source initiatives we have focused our attention on Nimbus, Eucalyptus and OpenNebula. We have also detailed endeavors in the research world that are relevant to the topic of interest and are not connected to any of the Cloud providers. By studying this report, the reader will be able to understand the role that each of the three Cloud features plays for a Cloud client, will understand their inner workings and will be aware of the state-of-the-art available in current commercial and open-source Cloud platforms.

Key-words: Cloud computing, auto-scaling, load balancing, monitoring

Gestion élastique des ressources, équilibrage de charge et monitoring dans les Clouds commerciaux et open-source

Résumé : Depuis ces dernières années, le phénomène du Cloud a eu une augmentation impressionnante de sa popularité à la fois dans l'industrie du logiciel et dans la recherche. Du point de vue du client, la fonctionnalité la plus intéressante que le Cloud fourni et le modèle d'allocation de ressources à la demande. Il permet aux plates-formes clientes d'augmenter le nombre de ressources pour pouvoir supporter un accroissement des requêtes et de réduire leur nombre lorsque la plate-forme a des ressources inutilisées et ceci à l'exécution. Grâce à cette fonctionnalité, les ressources physiques sont utilisées de manière plus efficace et le Client d'un Cloud réalise des économies. Réaliser ce type d'action n'est pas trivial et celà est réalisé en s'appuyant sur des fonctionnalités directes fournies par les Clouds. Trois de ces fonctionnalités sont la gestion élastique des ressources, l'équilibrage de charge et le monitoring. Elles représentent le corps de nos travaux présentés dans ce rapport de recherche. Ce rapport est divisé en trois parties principales, une pour chaque sujet étudié. Nous présentons chacun de ces sujets et présentons les détails de leur implémentation dans les Clouds commerciaux, académiques et open-source. Parmi les offres commerciales, nous nous sommes concentrés sur Amazon EC2, Microsoft Azure, GoGrid et RackSpace. En ce qui concerne les initiatives open-source, nous avons étudié Nimbus, Eucalyptus et OpenNebula. Nous avons également décrit les initiatives de recherche qui sont en relation avec les sujets traités mais pas forcement connectés à un des fournisseur de Cloud. En étudiant ce rapport, le lecteur sera capable de comprendre le rôle de chacune des fonctionalités pour un client de Cloud, de comprendre leur fonctionnement interne et il aura une vue globale de l'existant sur les plates-formes commerciales et publiques.

Mots-clés : Cloud computing, gestion élastique des ressources, équilibrage de charge, monitoring

Contents

1 Introduction

The increasing relevance of Cloud computing in the IT world is undeniable. Cloud providers have focused a lot of attention on providing facilities for Cloud clients, that make using the Cloud an easy task. These facilities range from automatic and configurable platform scaling and load balancing services to platform monitoring at different levels of granularity and configurable alert services. Given that there are no formal standards related to this topic, each Cloud provider has their own interpretation of the problem and their own way of addressing it.

In what follows, we will detail the topics of auto-scaling, load-balancing and monitoring. We will explore both Cloud provider and Cloud client points of view and examine what approaches are taken by the commercial providers and their open-source counterparts. Where an implementation is not available for one of the discussed Clouds, we will present alternative solutions by turning to available commercial services and open-source software. We will also present research work that has been done around the topic of interest.

The rest of this report is organized as follows: in Section 2 we present the notion of auto-scaling and detail its implementation in the current commercial and open-source Cloud providers, as well as in third party software. Section 3 presents the concept of load balancing related to the topic of Cloud platforms, its usefulness and its presence in the current available Cloud platforms and possible alternatives. The last of the three topics of interest, platform monitoring, is detailed in Section 4 and we conclude the current report in Section 5.

2 Cloud Auto-Scaling

Elasticity is regarded as one of the differentiating features of clouds. In fact, for some authors, it is considered the characteristic that makes clouds something other than "*an outsourced service with a prettier face*" [25]. Cloud users can quickly deploy or release resources as they need them, thus taking benefit of the typical pay-per-use billing model. They avoid potential over-provisioning of resources which implies investment in resources that are not needed. Also, increases on demand can be quickly attended to by asking the cloud for more resources, thus preventing a possible degradation of the perceived service quality.

However, to benefit from elasticity in typical Infrastructure-as-a-Service (IaaS) settings, the cloud user is forced to constantly control the state of the deployed system. This must be done in order to check whether any resource scaling action has to be performed. To avoid this, several auto-scaling solutions have been proposed by academia [26, 24] and by different cloud vendors. All these solutions allow users to define a set of scaling rules regarding the service hosted in the clouds. Each rule is composed by one or more conditions and a set of actions to be performed when those conditions are met. Conditions are typically defined using a set of metrics (which must be monitored by the cloud platform) like for example CPU usage, and some threshold. When the threshold is traversed then the condition is met.

Although cloud auto-scaling proposals all are based on this conditions + actions approach, they vary substantially in several aspects: which metrics are monitored (and so included in the rules definition); expressiveness of the con-

ditions defining mechanism; and which actions can be taken. Many of them focus on horizontal scaling, i.e., deploying or releasing VMs, while vertical scaling (like for example increasing physical resources of an overloaded server) is not considered, possibly due to the impossibility of changing the available CPU, memory, etc., on-the-fly in general purpose OSs.

Here we analyze the different auto-scaling solutions used by several cloud proposals, commercial ones such as Amazon EC2 and open source solutions such as Open Nebula. Also, we take a look at solutions developed by academia.

2.1 Auto-Scaling in Commercial Clouds

2.1.1 Amazon EC2

Amazon provides auto-scaling as part of the service offered by their IaaS EC2 public cloud. This service can be accessed by a web services API or through the command line. Auto-scaling in EC2 is based on the concept of *Auto Scaling Group* (ASG). A group is defined by:

- The configuration of the VMs that will be part of the group, where the configuration is given by the virtual image (that contains the OS and software stack of the VM) and hardware characteristics. As there it can only be an unique configuration per group, then all machines must by force have the same configuration. As a matter of fact each ASG represents a cluster, so this limitation can be assumed in many scenarios as real world clusters are often built by similar machines. But the lack of heterogeneous groups impedes certain useful configurations. For example, some users could benefit by replacing several "small" machines with one single "powerful" machine for cost reasons. Such replacement cannot be done automatically by EC2 auto-scaling service.

- Certain parameters such as the zone where VMs of the group will be deployed (among EC2's available regions, i.e. EU, US East...) or the minimum and maximum amount of VM instances allowed for the group. When setting a minimum size on the group, the user implicitly configures EC2 to automatically create a new VM whenever some of the running instances are shut down (e.g. because of a failure) and the minimum limit is exceeded.

Finally, the user can define a set of rules for each ASG. In EC2 jargon, the possible actions to be run are denoted *policies*. Each policy defines the amount of capacity (in absolute or relative values) to be deployed or released in a certain group. The platform will create or shut down VM instances in the ASG to meet that capacity demand. Triggers are denoted *metric alarms* and are based on the metrics served by EC2's monitoring service *CloudWatch* (see Section 4). Each metric alarm is defined by the metric and related statistic to be observed (like the average value), the evaluation period, and the threshold that will trigger the alarm. When the alarm is triggered, the action defined by the corresponding policy is run. Load balancers are automatically notified to start/stop sending requests to the created/stopped VMs.

2.1.2 Microsoft Azure

Microsoft Azure is considered a Platform-as-a-Service (PaaS) cloud as the Google App Engine platform or salesforce.com. PaaS clouds offer a runtime environment system (e.g. a servlets container) where users' components can be deployed and executed in a straightforward manner. Thus, PaaS clouds are said to offer an *additional abstraction level* when compared to IaaS clouds [29], so users do not have to handle virtual resources such as machines or networks to start running their systems. In such settings is the cloud system who must scale resources as needed by the container platform in a transparent manner, without any user intervention. Users are not required to monitor its service to scale resources, nor to define scalability rules.

Azure, nonetheless, is an exception. In Azure is the user who must configure the scalability settings, i.e. the platform does not handle resources scaling on behalf of the user. As part of the Azure cloud, there is a "Compute" service where users can request VMs instances, so Azure does not isolate users from resources, in contrast with other PaaS platforms. There is also available a *diagnostics mechanism* to obtain several metrics, and a *management API* to request certain actions on the deployed servers. But it is the user who must code the auto-scaling functionality she may require by using these APIs, i.e., Azure does not implement any embedded auto-scaling solution ready to be used by only configuring it, as for example EC2 does.

Not surprisingly, some commercial offers such as Paraleap [16] have emerged that try to address this severe limitation. Paraleap automatically scales resources in Azure to respond to changes on demand.

2.1.3 GoGrid

GoGrid does not implement any auto-scaling functionality. Similarly to Azure, it does provide an API to remotely command the addition or removal of VMs (servers), but it is up to the user to use this API method when required.

RightScale [18] is a *cloud management platform* that offers control functionality over the VMs deployed in different clouds. It provides auto-scaling functionality on top of GoGrid and EC2, based on *alerts* and associated *actions* to be run (one or more) each time an alarm is triggered. This is similar to the auto-scaling services of EC2. But there are some differences. First, alerts can be defined based not only on hardware metrics. Metrics regarding the state of software applications such as the Apache web server and MySQL engine are also available. Also, several actions can be associated to the alert, like for example sending emails to administrators. Besides, these actions can be run periodically for defined intervals of time, not just once. Finally, alerts and actions are usually defined at the server level. But scaling actions are an exception: they are performed only if a certain user-defined percentage of servers "vote" for the action to be run.

2.1.4 RackSpace

As in the case of GoGrid, RackSpace has not built in auto-scaling capabilities, although it does provide an API for remote control of the hosted VMs. Thus, the user is responsible for monitoring the service and taking the scaling decisions. The creation and removal of resources is done through calls to the remote API.

Enstratus [4] is a *cloud management platform* that offers control functionality over the VMs deployed in different clouds, as RightScale does. It provides auto-scaling functionality on top of all those clouds, including RackSpace, again very similar to the one provided by the auto-scaling services of EC2.

RightScale [18] has recently added support for VMs deployed on RackSpace.

Scalr [19] is an open source project that handles scaling of cloud applications on EC2 and RackSpace. It only manages web applications based on Apache and MySQL, and the scaling actions are decided by a built-in logic. Users cannot configure how their applications must scale in response to changes on load.

2.2 Implementations of Auto-Scaling in Open-Source Clouds

2.2.1 Nimbus

Nimbus does not have any embedded auto-scaling functionality, it is a tool to create clouds from datacenter resources. In [24] the authors introduce a *resource manager* able to ask for resources from third-party clouds when the local resources are under high demand. The decisions about when to scale the deployment by adding new VMs is taken by *policies*, implemented as Python modules. These policies are the equivalent to the elasticity rules present in other systems.

2.2.2 Eucalyptus

In Eucalyptus there is no out-of-the-box auto-scaling functionality. Eucalyptus is focused on the management at virtual resource level, and does not control the services running on it. Hence it cannot be aware of their state and so it cannot decide when to add or release resources to the service.

Makara [8] offers a "wrapper" service for the deployment and control of services running in the cloud, an approach similar to RightScale, but aimed at Java and PHP-based applications. Makara can manage services running on top of one or many cloud providers, including EC2 and private clouds based on Eucalyptus. The auto-scaling functionality in Makara allows to define when to create or release VMs using thresholds over two metrics: CPU, and requests to node.

Scalr can also be used for applications running on Eucalyptus clouds.

2.2.3 OpenNebula

OpenNebula does not provide auto-scaling. It promotes a clear distinction of the roles at each layer of the cloud stack. Open Nebula, which would be at the bottom of this stack, aims to ease the management of the virtual infrastructure demanded, and it assumes that scaling actions should be controlled and directed at a higher level. I.e., OpenNebula is not "aware" of the services it hosts, or of their state. Hence, OpenNebula is not in charge of supervising the deployed systems.

The work in the "*OpenNebula Service Management Project*" [15] tries to develop a component to be run on top of OpenNebula that handles services (understood as a cluster of related VMs) instead of raw hardware resources. Support for auto-scaling could be added in the future. Most mature seems Claudia, discussed in Section 2.3.

2.3 Claudia: A Proposal for Auto-Scaling

Auto-scaling is getting some attention from the research community, as the limitations of the present solutions become more and more clear. One of such limitations is the lack of flexibility. The auto-scaling configuration is set using a GUI with limited choices regarding the metrics to be monitored and the conditions that will trigger the corresponding scaling actions.

In this regard, Claudia [26] proposes a more flexible solution. Claudia is a Cloud tool designed to handle not just VMs, but whole services which in turn are usually composed of a set of connected VMs. Claudia's declared goal is to offer cloud users an abstraction level closer to the services' lifecycle, that does not force users to deal with single VMs. As part of the service definition the user will describe the VMs that must be instantiated and the dependencies among them. Claudia will deploy the service following the instructions given by the user, including the instantiation order of VMs. Also, it will make available to the VMs the configuration information they require, including the one that will be only available at deployment time. For example, a webserver could need the IP address to connect to a database server which is part of the same service, which is only known once the database has been started. Such dependencies are managed by Claudia (such a configuration process is also denoted *contextualization*).

Also, as part of this service-level handling, Claudia is in charge of scaling the resources associated to the service. As part of the service definition (format based on an extension to the *Open Virtualization Format*) users can include *elasticity rules* based on arbitrary service level metrics (such as the number of transactions). The service will forward these values to Claudia through the monitoring system. Claudia uses a rule engine to constantly check the conditions defined for each rule and trigger the corresponding actions when the conditions are met. Claudia was built and tested as a set of components running on top of OpenNebula, but is not tied to any particular cloud platform. The Claudia proposal goes a step further than other auto-scaling systems. First, scaling decisions can be based on *any* metric, while other auto-scaling proposals restrict the metrics that can be used to define the scalability rules that will govern the service. Therefore, scaling can be applied to any service regardless of the metrics that represent its state. Also, rules are defined using the extensions that Claudia proposes to the Open Virtualization Format [22] (format to define the deployment of collections of VMs, promoted by the DMTF), that have a richer expressiveness than typical rule definition schemes in present cloud offers. Finally, each rule can have many actions associated (not just one).

2.4 Implementing and Auto-Scaling Cloud System

To conclude this section, we briefly enumerate the three main functionalities to be implemented in a cloud system to enable it to scale the software it hosts:

- *Monitoring* (see Section 4) is required to get information about the state of the running services and the resources they use. Two challenges are involved: first, being able to handle metrics from up to thousands of services, each one running on several VMs; second adding arbitrary service-level metrics that represent the state of the deployed software beyond simple hardware-level metrics.

- *Load-balancing* (see Section3) mechanisms should be ready to update the set of available server replicas, as it can change at any moment due to the creation and removal of VM instances.
- *Rule checking* is in charge of reading the metrics values to verify when the conditions that trigger the scaling actions are met.

3 Cloud Client Load Balancing

This concept of load balancing is not typical to Cloud platforms and has been around for a long time in the field of distributed systems. In its most abstract form, the problem of load balancing is defined by considering a number of parallel machines and a number of independent tasks, each having its own load and duration [21]. The goal is to assign the tasks to the machines, therefore increasing their load, in such a way as to optimize an objective function. Traditionally, this function is the maximum of the machine loads and the goal is to minimize it. Depending on the source of the tasks, the load balancing problem can be classified as: *offline load balancing* where the set of tasks is known in advance and cannot be modified and *online load balancing* in the situation that the task set is not known in advance and tasks arrive in the system at arbitrary moments of time.

In the case of Cloud computing we can consider load balancing at two different levels: Cloud provider level and Cloud client level. From the point of view of the Cloud provider, the load balancing problem is of type online and is mapped in the following way:

- The parallel machines are represented by the physical machines of the Cloud's clusters
- The tasks are represented by client requests for virtual resources
- Cloud client requests can arrive at arbitrary moments of time

In the case of Cloud client's virtual platform, the load balancing problem is mapped in the following way:

- The parallel machines translate into the virtual resources that the Cloud client has currently running
- The tasks translate into client requests to the Cloud client's platform
- End user requests can arrive at arbitrary moments of time

As a consequence, in Cloud platforms, load balancing is an online problem where end user requests that enter the Cloud client's application need to be distributed across the Cloud client's instantiated virtual resources with the goal of balancing virtual machine load or minimizing the number of used virtual machines.

Although load balancing is not a unique feature to Cloud platforms, it should not be regarded as independent from auto-scaling. In fact, the two need to work together in order to get the most efficient platform usage and save expenses.

The end goal of load balancing from the Cloud client's point of view is to have a more efficient use of the virtual resources that he has running and

thus reduce cost. Since most Cloud providers charge closest whole hour per virtual resource, then the only way that cost saving is achieved is by reducing the number of running virtual resources, while still being able to service client requests. It follows that load balancing should be used in conjunction with auto-scaling in order to reduce cost. As a result we have the following usage scenarios:

1. **high platform load** when the Cloud client's overall platform load is high, as defined by the Cloud client. The platform needs to scale up by adding more virtual resources. The load balancing element automatically distributes load to the new resources once they are registered as elements of the platform and therefore reduces platform load.

2. **low platform load** when the Cloud client's overall platform load is low, as defined by the Cloud client. In this situation, the platform needs to scale down by terminating virtual resources. This is not as trivial as the previous scenario, because the load balancing element typically assigns tasks to all resources and therefore prevents resources from reaching a state where their load is zero and can be terminated. In this situation, the load balancing element needs to stop distributing load to the part of the platform that will be released and, even more, the currently-running tasks of this part of the platform need to be migrated to ensure that a part of the platform will have zero load and therefore can be released.

Load balancing also brings some issues as side effects along with it. One of these is session affinity. Because load balancers distribute load evenly among available nodes, there is no guarantee that all the requests coming from one user will be handled by the same node from the pooled resources. This has the implication that all context related to the client session is lost from one request to another. This is usually an undesired effect. In the great majority of situations, it is desired that requests from the same client be handled by the same node throughout the duration of the client's session. In modern clouds this is referred to as session stickiness.

Mapping of virtual resources to physical resources also has an impact on Cloud clients. There is usually a compromise between the following two opposite use cases:

- The Cloud provider achieves a more efficient resource usage by trying to minimize the number of physical hosts that are running the virtual resources. The downside for the Cloud client is the fact that his platform is at a greater risk in case of hardware failure because the user's virtual resources are deployed on a small number of physical machines.

- The virtual resources are distributed across the physical resources. Thus the risk of failure is less for Cloud clients in case of hardware failure. On the downside, there is a greater number of physical machines running and thus more power usage.

3.1 Load Balancing in Commercial Clouds

The problem of load balancing in all Cloud platforms may be the same, but each Cloud provider has its own approach to it, which is reflected in the services they offer and their differences with respect to other providers.

3.1.1 Amazon EC2

Amazon EC2 offers load balancing through their *Amazon Elastic Load Balancing* service [2]. The Cloud client can create any number of *LoadBalancer*s. Each *LoadBalancer* will distribute all incoming traffic that it receives for its configured protocol to the EC2 instances that are sitting behind the *LoadBalancer*. One single *LoadBalancer* can be used to distribute traffic for multiple applications and across multiple *Availability Zone*s, but limited to the same Amazon EC2 *Region*.

If an instance that is behind the *LoadBalancer* reaches an unhealthy state, as defined by the *LoadBalancer*'s health check, then it will not receive any new load, until its state is restored so that it passes the health check. This feature increases the fault tolerance of Cloud client applications by isolating unhealthy components and giving the platform notice to react.

Amazon's *Elastic Load Balancing* service has two ways of achieving stickiness:

1. A duration-based sticky session in which case the load balancers themselves emit a cookie of configurable lifespan, which determines the duration of the sticky session.

2. An application-controlled sticky session in which case the load balancers are configured to use an existing session cookie that is completely controlled by the Cloud client's application.

Related to sticky sessions, it is worth noting that Amazon EC2 does not support load balancing for HTTPS traffic. This is due to the fact that the cookies are stored in the HTTP header of the request and all HTTPS traffic is encrypted. So to perform session stickiness when load balancing HTTPS traffic, the balancers would need to have the application's SSL certificate, which cannot be done in the current version of the load balancers.

As for pricing, the Cloud user is charged for the running time of each *LoadBalancer*, rounded up to an integer number of hours, and also for the traffic that goes through the *LoadBalancer*. Pricing for load balancers is calculated identically to pricing for any other instance type, given that the balancers are not hardware, but regular instances configured to work as load balancers.

3.1.2 Microsoft Azure

In Windows Azure, Microsoft has taken an automatic approach to the load balancing problem. Load is automatically distributed among available work resources by using a round robin algorithm in a way transparent to the platform's users [10].

Microsoft's Windows Azure offers a Cloud middleware platform for managing applications. This middleware is known under the name of *AppFabric* [9]. Essentially, this is a managed PaaS Cloud service. For Cloud clients, the AppFabric service abstracts away deployment and service management. The control unit for the AppFabric service is the *AppFabric Controller*.

Load balancing for applications running under the AppFabric service is achieved by using hardware load balancers. The load balancers have redundant copies to reduce failure.

The load balancing mechanism is aware of the health state of the nodes that it balances load to (in the Azure platform, these nodes are called roles and can be either of type *Web role* or *Worker role*). The *AppFabric Controller* maintains an updated list of health states for the nodes in the platform. This is done in two ways: the node itself communicates a bad health state to the controller, or the controller queries the node for its health (usually by pinging it). When a node becomes unhealthy, traffic will cease to be distributed to it by the load balancers, until its health state is restored to normal.

The *AppFabric Controller* has fault tolerance mechanisms when it comes to virtual resource mapping to physical resources. This is done by introducing the concept of *fault domains*. A fault domain represents a point of hardware failure and in Azure it is actually a collection of physical network racks. When a physical failure happens inside a fault domain, it is very probable that all the virtual resources running in that fault domain will be affected.

A Cloud client application has a number of fault domains associated to it. They are controlled by the *AppFabric Controller* and cannot be changed by the Cloud client. The default number of fault domains per application is 2.

3.1.3 GoGrid

With respect to load balancing, GoGrid uses redundant f5 hardware load balancers [7]. Each account has free usage of the load balancers.

The load balancers can be configured in terms of what load balancing algorithm to use. The user has a choice between two available approaches:

1. Round robin: with this configuration, traffic is balanced evenly among available pooled nodes.
2. Least connect: this configuration makes the load balancers send new traffic to the pooled node with the least number of currently active concurrent sessions.

Load balancing can disturb client sessions if traffic for the same session is not routed to the same server node that initiated the session throughout the whole duration of the session. To prevent this, load balancers can be configured with a persistency option. The user can choose one of the following three:

1. None: in which situation, traffic is distributed as according to the balancing algorithm selected, ignoring possible session problems.
2. SSL Sticky: in which situation, all SSL traffic is routed to the same destination host that initiated the session. When a new SSL session is initiated, the destination node for handling the first request is chosen based on the balancing algorithm selected for the load balancer.
3. Source address: in which situation, all traffic from a source address is routed to the same destination node after the initial connection has been made. The destination node for handling the first connection of a new source is chosen based on the algorithm that the load balancer is configured to use.

The load balancers also check the availability of nodes in the balancing pool. If one node becomes unavailable, the load balancer removes it from the pool automatically.

3.1.4 Rackspace

Rackspace Cloud offers two types of Cloud services: *Cloud Servers* and *Cloud Sites*. The *Cloud Servers* service is of type IaaS in which all auto-scaling, load balancing and backup related issues are left in the hands of the Cloud client.

On the other hand, the *Cloud Sites* service targets automated scaling, load balancing and daily backups [20]. Thus it leverages the benefits of Clouds without a need for too much interaction from the clients. In fact, the Cloud clients do not see the virtual resources that they are currently using. The Cloud platform is presented in a very abstract manner, the Cloud client deploys his application to the Cloud without having intimate knowledge of the Cloud's underlying scaling, balancing, and backup platforms.

The algorithm used for distributing the load is round robin.

Pricing is done based on the client's platform usage in terms of disk space, bandwidth and compute cycle, which is a unit that enables Rackspace to quantify their platform's computational usage. Users are not charged for use of load balancing service.

3.2 Implementations of Load Balancing in Open-Source Clouds

3.2.1 Nimbus

From the Cloud provider's point of view, there is a feature still under development in Nimbus that allows back-filling of partially used physical nodes . This will also allow preemptable virtual machines, an identical concept to Amazon EC2's spot instances.

From the virtual platform level, there is ongoing work for a high-level tool that monitors virtual machine deployment and allows for compensation of stressed workloads based on policies and sensor information .

3.2.2 Eucalyptus

Eucalyptus does not contain an implicit load balancing service for low-level virtual machine load balancing or high-level end-user request load balancing. Nor does Eucalyptus have a partnership program similar to RackSpace's Cloud Tools or GoGrid's Exchange programs.

As alternatives, one can opt for a complete managed load balancing solution offered by third party providers. Given that Eucalyptus implements the same management interface as Amazon EC2 does, it is relatively easy to find such commercial services.

3.2.3 OpenNebula

OpenNebula is service agnostic. This means that the service being deployed on OpenNebula needs to take care of load balancing on its own.

From a virtual resource balancing point of view, OpenNebula's virtual resource manager [14] is highly configurable. Each virtual machine has its own placement policy and the virtual resource manager places a pending virtual machine into the physical machine that best fits the policy. This is done through the following configuration groups:

- The *Requirements* group is a set of boolean expressions that provide filtering of physical machines based on their characteristics.

- The *Rank expression* group is a set of arithmetic statements that use characteristics of the physical machines and evaluate to an integer value that is used for discriminate between the physical machines that have not been filtered out. The physical host with the highest rank is the one that is chosen for deploying the virtual machine.

To choose the best physical machine is done by first filtering based on the requirements of the virtual machine and then choosing the physical machine with the highest rank for deployment.

It is trivial to obtain a policy that minimizes the number of used physical resources. It is also possible to obtain a policy that achieves a good distribution of virtual machines among the physical machines with the goal of minimizing the impact that a hardware failure would have on a Cloud client's platform.

The virtual machine placement policies can be configured per virtual machine instance; however, when using a Cloud interface, this cannot be specified by the user and so they are defined by the Cloud administrator per virtual machine type.

4 Cloud Client Resource Monitoring

Keeping track of the platform health is crucial for both the platform provider and the platform user. This can be achieved by using platform monitoring systems. Monitoring can be done on two different levels, depending on the beneficiary of the monitoring information:

1. Low-level platform monitoring is interesting from the point of view of the platform provider. Its purpose is to retrieve information that reflects the physical infrastructure of the whole Cloud platform. This is relevant to the Cloud provider and is typically hidden from the Cloud clients, as their communication to the underlying hardware goes through a layer of virtualization. In general, it is the responsibility of the Cloud provider to ensure that the underlying hardware causes no visible problems to the Cloud clients. For commercial Cloud providers, the low-level monitoring service is usually kept confidential.

2. High-level monitoring information is typically interesting for Cloud clients. This information is focused on the health of the virtual platform that each individual Cloud client has deployed. It follows that the Cloud providers have little interest in this information, as it is the up to the client to manage his own virtual platform as he sees fit. Due to privacy constraints, platform monitoring information is only available to the virtual platform owner and is hidden from the other Cloud clients.

Although this separation is intuitive, there is no clear separation between the two. Each Cloud provider comes with its own interpretation and implementation of resource monitoring.

4.1 Monitoring in Commercial Clouds

4.1.1 Amazon EC2

As a commercial Cloud, the low-level monitoring system that Amazon uses for acquiring information on its physical clusters is kept confidential.

The approach that Amazon EC2 has taken with respect to high-level resource monitoring is to provide a service called *CloudWatch* [1] that allows monitoring of other Amazon services like EC2, Elastic Load Balancing and Amazon's Relational Database Service. The monitoring information provided to a Cloud client by the *CloudWatch* service is strictly related to the Cloud client's virtual platform.

The *CloudWatch* service collects the values of different configurable measurement types from its targets and stores them implicitly for a period of two weeks. This period of two weeks represents the expiration period for all available measures and is, in essence, a history of the measure that allows viewing of the evolution in the measurements. *CloudWatch* is actually a generic mechanism for measurement, aggregation and querying of historic data. All the measurements are aggregated over a period of one minute.

In association with the Elastic Load Balancer service and the Auto-scaling feature, *CloudWatch* can be configured to automatically replace platform instances that have been considered unhealthy, in an automatic manner.

CloudWatch comes with an alarm feature. An alarm has a number of actions that are triggered when a measure acquired by the monitoring service increases over a threshold or decreases under a threshold. The measures are configurable and the thresholds correspond to configurable limits for these measures. The possible actions are either a platform scaling action or a notification action. In the case of notification, there are a number of possible channels for doing this. They include Amazon's SNS and SQS services, HTTP, HTTPS or email. The actions are executed only when a measure transitions from one state to another and will not be continuously triggered if a measure persists on being outside the normal specified working interval.

Related to pricing, the *CloudWatch* service is charged separately with a single price per hour, regardless of the resource that is being monitored. Recently, Amazon changed the basic monitoring plan to be free of charge. This includes collection of values every five minutes and storage of these values for a period of two weeks. A detailed monitoring plan is also available that offers value collection at a rate of once per minute and is charged per hour of instance whose resource values are collected.

4.1.2 Microsoft Azure

Information about the monitoring system used for low-level platform monitoring of the whole Azure platform has not been given. However, the approaches that the Azure Cloud client has to application monitoring have been documented [11].

For monitoring applications deployed on Microsoft Windows Azure, the application developer is given a software library that facilitates application diagnostics and monitoring for Azure applications. This library is integrated into the Azure SDK. It features performance counters, logging, and log monitoring.

Performance counters are user-defined and can be any value related to the Cloud application that is quantifiable.

The logging facilities of the library allow tapping into:

- Application logs dumped by the application. This can be anything that the application developer wants to log.
- Diagnostics and running logs
- Windows event logs that are generated on the machine that is running a worker role
- IIS logs and failed request traces that are generated on the machine that is running a web role
- Application crash dumps that are automatically generated upon an application crash

The storage location for the log files is configurable. Usually one of two storage environments is used: local storage or Azure storage service. The former is a volatile storage that is included in the virtual machine's configuration, while the latter is a storage service offered by Azure and has no connection to the virtual machine's storage. Usually the latter is preferred for what the Cloud user considers to be permanent logs while the former is used as a volatile storage.

There is no automatic monitoring mechanism for web roles and worker roles running on Microsoft Azure.

The cost implications of using the diagnostics and monitoring libraries are only indirect. There is no fee associated to using them, but there is a fee for storing information in a non-volatile persistence storage service and also in querying that storage service.

4.1.3 GoGrid

There is currently no public information related to how GoGrid achieves low-level monitoring on their platform.

GoGrid features a collaboration program. This program runs under the name of *GoGrid Exchange* [6] and presents third-party services that can prove useful to Cloud clients. These services include third-party packages that target monitoring features ranging from platform security monitoring to resource usage monitoring and database monitoring. These services also include the possibility of configurable alerts based on the values of the monitored measures.

4.1.4 RackSpace

As in the case of the other commercial Cloud providers, the approach used by RackSpace for low-level platform monitoring is not public. In what follows we will detail how Cloud clients can monitor their platform on RackSpace.

The Rackspace *Cloud Sites* service offers monitoring capabilities at the whole application level for fixed parameters that include used compute cycle count, used bandwidth and storage. This fits well into the usage scenario that *Cloud Sites* offer: that of a PaaS service; but lack of finer-grained sub-application level monitoring can be a downside for some Cloud clients. Again at an application level, logging for applications deployed on *Cloud Sites* is offered, but in a per-request manner.

On the other hand, the *Cloud Servers* service, which is an IaaS-type of service, does also have monitoring capabilities through the use of third-party partner software, especially tailored for Rackspace's *Cloud Servers* service. These partner solutions are aggregated by Rackspace under the name of *Cloud Tools* [17]. Among these partner services, one can find complete monitoring solutions ranging from general virtual machine monitoring to specialized database monitoring. The services that are specialized on monitoring also feature configurable alert systems.

Recently, RackSpace has acquired CloudKick [3], a multi-cloud virtual platform management tool. CloudKick has a broad range of monitoring features for virtual machines. These include different monitoring metrics from low-level metrics like CPU / RAM / disk utilization to high-level metrics like database statistics, HTTP / HTTPS and others. The monitoring metrics can be extended by custom plugins that are able to monitor anything that the user defines. Measured data can be presented in raw form or aggregated by user-defined means. For data visualization, a real-time performance visualization tool is also provided.

CloudKick also features alerts that have a configurable trigger and repeat interval. The alert prompt can be sent by SMS, email or HTTP.

4.2 Implementations of Monitoring in Open-Source Clouds

4.2.1 Nimbus

Nimbus features a system of Nagios[12] plugins that can give information on the status and availability of the Nimbus head node and worker nodes, including changes of the virtual machines running on the worker node.

Also, there is active work being done around building a higher level tool that is able to monitor deployed virtual machines and compensate for stress points by using monitor information from sensors and configurable policies.

4.2.2 Eucalyptus

Since version 2.0, Eucalyptus has introduced monitoring capabilities[5] for the running components, instantiated virtual machines and storage service. This is done by integrating Eucalyptus monitoring into an existing and running monitoring service. Currently, monitoring has been integrated with Ganglia[27] and Nagios. In Eucalyptus this is done by means of scripts that update the configuration of the running monitoring service to also monitor Eucalyptus components and virtual machines.

As alternative solutions to achieving monitoring at a hardware level, one can employ one of the monitoring systems that have been designed and used in grid environments. Some such systems have been detailed in Section 4.3.

We can also opt for a completely managed monitoring solution offered by third party providers. Given that Eucalyptus implements the same management interface as Amazon EC2 does, it is relatively easy to find such commercial services. Among the commercial solutions available, we can enumerate the following as compatible with Eucalyptus (but the list is not exhaustive):

RightScale offers a Cloud management environment for Cloud clients. This management environment also includes a real-time platform monitoring

sub-service.

enSTRATUS provides a set of tools for the provisioning, managing and monitoring of Cloud infrastructures with applicability to private and public Cloud platforms.

Makara offers a PaaS service on top of an IaaS Cloud. They focus on the deployment, management, scaling and monitoring of PHP and Java applications deployed on Clouds. With respect to monitoring, Makara supports real-time and historical performance monitoring.

4.2.3 OpenNebula

The built-in monitoring capabilities of OpenNebula focus on the Cloud provider's interest in the physical resources. This functionality is found in the OpenNebula module called the *Information Manager* [13].

The Information Manager works by using *probes* to retrieve information from the cluster's nodes. The probes are actually custom scripts that are executed on the physical nodes and output pairs of Attribute=Value on their standard output. The pairs are collected and centralized. As a requirement, the physical nodes should be reachable by SSH without a password.

Currently, the probes are focused on retrieving only information that underlines the state of the physical nodes and not its running virtual machines (CPU load, memory usage, host name, hypervisor information, etc.). It is advised that this information not be mixed with information of interest to the Cloud client. For such a task, the OpenNebula community recommends using a service manager tool that is a separate entity from OpenNebula. As possible solutions, we can consider commercial services that are specialized in Cloud platform management, including monitoring. Such solutions have been described in the previous sections. Alternatively, we can also turn to cluster monitoring solutions that come from the open-source world, some of which are the result of long research endeavors and have been described in Section 4.3.

While still under development, the next version of the Information Manager is based on the Ganglia multi-cluster monitoring tool.

4.3 Other Research Endeavors That Target Monitoring in Large-Scale Distributed Systems

Over the years as grid computing evolved, so did the need for monitoring large-scale distributed platforms that are built on top of grids. There have been many fruitful research efforts for designing and implementing monitoring systems for large-scale platforms. In the following, we will highlight some of these efforts. The list of research projects that we present is not exhaustive for the field of large-scale platform monitoring.

4.3.1 The Network Weather Service - NWS

NWS [30] has the goal of providing short-term performance forecasts based on historic performance measurements by means of a distributed system. To achieve this, NWS has a distributed architecture with four different types of component processes:

Name Server process is responsible for binding process and data names with low level information necessary when contacting a process

Sensor process is responsible for monitoring a specified resource. The first implementation contained sensors for CPU and network usage. Sensors can be added dynamically to the platform.

Persistent state process is responsible for storing and retrieving monitoring data. By using this type of process, the process of measuring is disconnected from the place where measurements are stored.

Forecaster process is responsible for estimating future values for a measured resource based on the past measure values. The forecaster applies its available forecasting models and chooses the value of the forecaster with the most accurate prediction over the recent set of measurements. This way, the forecaster insures that the accuracy of its outputs is at least as good as the accuracy of the best forecasting model that it implements.

To increase fault tolerance, NWS uses an adaptive and replicated control strategy by an adaptive time-out discovery and a distributed leader election protocol. Sensors are grouped into hierarchical sets called *cliques*. A Sensor can only perform intra-clique measurements, thus limiting contention and increasing scalability of the system.

The implementation uses TCP/IP sockets because they are suited for both local area and wide area networks and they provide robustness and portability.

4.3.2 Ganglia

Ganglia [27] addresses the problem of wide-area multi-cluster monitoring. To achieve this it uses a hierarchy of arbitrary number of levels with components of two types:

Gmon component responsible for local-area monitoring. To gather information from cluster nodes, Gmon uses multicast over UDP, which has proved to be an efficient approach in practice, and it also makes Ganglia immune to cluster node joins and parts.

Gmeta component is responsible for gathering information from one or more clusters that run the Gmon component or from a Gmeta component running in a lower level of the tree hierarchy. Communication between the two components is done by using XML streams over TCP.

In order to achieve almost linear scalability with the total number of nodes in the clusters, the root Gmeta component should not be overwhelmed with monitoring data. To do this, an 1-level monitoring hierarchy should be avoided. Instead, an N-level monitoring tree hierarchy should be deployed, where it is dependent on the number of nodes in the cluster. There is a limitation here in the sense that although nodes can be dynamically added and removed from a cluster without needing to manually update the Ganglia hierarchy, the same cannot be said for Gmon and Gmeta components. The Gmeta needs to have *a priori* knowledge of the of its underlying child nodes.

4.3.3 Supermon

Supermon [28] aims at providing a high-speed cluster monitoring tool, focusing on a fine-grained sampling of measurements. To achieve this, Supermon uses three types of components:

Kernel module for monitoring provides measurements at a high sampling rate. Values are represented in the form of s-expressions.

Single node data server (mon) is installed for each monitoring kernel module. It parses the s-expressions provided by the kernel module. For each client connected to this server, it presents measurement data filtered by the client's interest. Data is sent by means of TCP connections.

Data concentrator (Supermon) gathers data from one or several mon or Supermon servers. They also implement the same per client filtering capability that mon servers have. Hierarchies of Supermon servers are useful to avoid overloading a single Supermon, especially in situations where a large number of samples is required or there is a large number of nodes that are monitored.

4.3.4 RVision

RVision (Remote Vision) [23] is an open tool for cluster monitoring. It has two basic concepts that make it highly configurable:

Monitoring Sessions are actually self-contained monitoring environments. They have information on what resource to monitor and what acquisition mechanism to use for the monitoring process associated to the resource.

Monitoring Libraries are actually collections of routines that are used for resource measurement information acquisition. These routines are dynamically linked at runtime and thus information acquisition, which is occasionally intimately tied to the resource that is being measured, is disconnected from the general mechanisms of monitoring.

The architecture of RVision is a classical master-slave architecture. The master node is represented by the RVCore. It is responsible for managing all the active sessions and distributing the monitoring information to the clients. The communication layer is implemented by using TCP and UDP sockets.

The slave part corresponds to the RVSpy component. There is one such component running on each node of the cluster that is to be monitored. The monitoring libraries that are needed for information acquisition are dynamically linked to the RVSpy. The slave component communicates its acquired information to the master by means of UDP, as this is a low-overhead protocol, and cluster nodes are usually connected by means of a LAN, where package loss is usually small.

5 Conclusions

The current work aims to familiarize the reader with the importance of auto-scaling, load balancing and monitoring for a Cloud client platform. The elasticity of Cloud platforms is reflected in their auto-scaling feature. This allows

Cloud client platforms to scale up and down depending on their usage. Achieving automatic client platform scaling is done in different ways, depending on the Cloud platform itself. One can opt for the Cloud's built-in auto-scaling feature if present in Amazon EC2, Microsoft Azure, or use a third party Cloud client management platform that offers this functionality: RightScale, Enstratus, Scalr, that are usable in most Cloud platforms. GoGrid and RackSpace have partner programs the GoGrid Exchange and the RackSpace Cloud Tools that provide custom-made tools that work on top of the hosting service that they offer.

Load balancing has the goal of uniformly distributing load to all the worker nodes of the Cloud client's platform. This is done by means of an entity called a load balancer. This can be either a dedicated piece of hardware that is able to distribute HTTP/HTTPS requests between a pool of machines the case of Microsoft Azure, GoGrid, or with a Virtual Machine instance that is configured by the platform provider or by the client himself to do the same job the case of Amazon EC2, Rackspace, Nimbus, Eucalyptus and OpenNebula.

In order to make sure that the platform is in a healthy working state, platform monitoring is used. This is done by means of a service that periodically collects state information from working virtual machines. The monitoring service can either be built as a part of the Cloud platform, as is the case of Amazon's CloudWatch, Microsoft Azure's monitoring package, Nimbus' Nagios plugins, Eucalyptus 2.0's monitoring service and OpenNebula's Information Manager. GoGrid and RackSpace offer this feature by means of their partner programs. Alternatively, Cloud clients can always choose a third party monitoring solution. As examples, we can enumerate CloudKick, Makara or any of the third party Cloud client platform management tools presented above.

Ultimately, all the three presented features are designed to work hand-in-hand and have the high-level goal of ensuring that the Cloud client's platform reaches a desired QoS level in terms of response time and serviced requests, while keeping the cost of running the platform as low as possible.

The readers of the current report should now have a clear understanding of the importance of the three main topics discussed here with respect to Cloud client platforms. We have presented the working principles of the three topics, along with a survey on the current available commercial and open-source Clouds. We hope that the current work can help current and future Cloud clients in making informed decisions.

References

[1] Amazon web services cloud watch Web Site, November 2010.

[2] Aws elastic load balancing Web Site, November 2010.

[3] Cloudkick Web Site, November 2010.

[4] Enstratus Web Site, December 2010.

[5] Eucalyptus monitoring Web Site, November 2010.

[6] Gogrid exchange Web Site, November 2010.

[7] Gogrid load balancing service Web Site, November 2010.

[8] Makara Web Site, December 2010.

[9] Microsoft azure appfabric Web Site, November 2010.

[10] Msdn forum - microsoft azure load balancing service Web Site, November 2010.

[11] Msdn library - azure monitoring and diagnostics Web Site, November 2010.

[12] Nagios project Web Site, November 2010.

[13] Opennebula information manager Web Site, November 2010.

[14] Opennebula scheduling policies Web Site, November 2010.

[15] OpenNebula Sservice Management Project, December 2010.

[16] Paraleap windows azure dynamic scaling Web Site, November 2010.

[17] Rackspace cloud tools Web Site, November 2010.

[18] RightScale Web Site, December 2010.

[19] Scalr Web Site, December 2010.

[20] Rackspace cloud sites load balancing service Web Site, January 2011.

[21] T.C.K. Chou and J.A. Abraham. Load balancing in distributed systems. *Software Engineering, IEEE Transactions on*, SE-8(4):401 – 412, 1982.

[22] S. Crosby, R. Doyle, M. Gering, M. Gionfriddo, S. Grarup, S. Hand, M. Hapner, D. Hiltgen, M. Johanssen, L.J. Lamers, J. Leung, F. Machida, A. Maier, E. Mellor, J. Parchem, S. Pardikar, S.J. Schmidt, R.W. Schmidt, A. Warfield, M.D. Weitzel, and J. Wilson. Open Virtualization Format Specification (OVF). Technical Report DSP0243, 2009.

[23] Tiago C. Ferreto, Cesar A. F. de Rose, and Luiz de Rose. Rvision: An open and high configurable tool for cluster monitoring. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:75, 2002.

[24] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic site: Using clouds to elastically extend site resources. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:43–52, 2010.

[25] Dustin Owens. Securing elasticity in the cloud. *Queue*, 8(5):10–16, 2010.

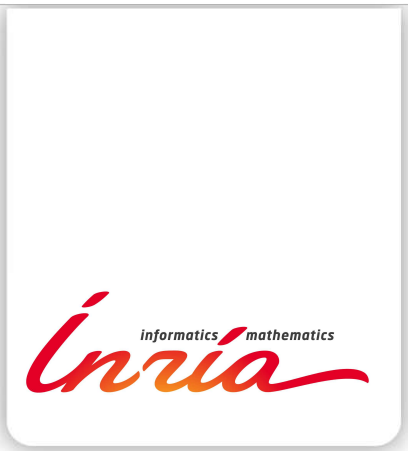
[26] Luis. Rodero-Merino, Luis M. Vaquero, Víctor Gil, Fermín Galán, Javier Fontán, Rubén Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26:1226–1240, October 2010.

[27] Federico D. Sacerdoti, Mason J. Katz, Matthew L. Massie, and David E. Culler. Wide area cluster monitoring with ganglia. *Cluster Computing, IEEE International Conference on*, 0:289, 2003.

[28] Matthew J. Sottile and Ronald G. Minnich. Supermon: A high-speed cluster monitoring system. *Cluster Computing, IEEE International Conference on*, 0:39, 2002.

[29] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[30] Rich Wolski, Neil T Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757 – 768, 1999.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399