Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Joint elastic cloud and virtual network framework for application performance-cost optimization

Tram Truong Huu · Guilherme Koslovski ·
Fabienne Anhalt · Johan Montagnat ·
Pascale Vicat-Blanc Primet

Received: date / Accepted: date

Abstract Cloud computing infrastructures are providing resources on demand for tackling the needs of large-scale distributed applications. To adapt to the diversity of cloud infrastructures and usage, new operation tools and models are needed. Estimating the amount of resources consumed by each application in particular is a difficult problem, both for end users who aim at minimizing their costs and infrastructure providers who aim at controlling their resources allocation. Furthermore, network provision is generally not controlled on clouds. This paper describes a framework automating cloud resources allocation, deployment and application execution control. It is based on a cost estimation model taking into account both virtual network and nodes managed by the cloud. The flexible provisioning of network resources permits the optimization of applications performance and infrastructure

Tram Truong Huu
University of Nice - Sophia Antipolis
I3S Laboratory, FRANCE
Tel.: +33 4 92 96 50 58
E-mail: tram@polytech.unice.fr
http://modalis.polytech.unice.fr/~tram

Guilherme Koslovski
INRIA - University of Lyon, FRANCE
E-mail: guilherme.koslovski@ens-lyon.fr
http://perso.ens-lyon.fr/guilherme.koslovski

Fabienne Anhalt
INRIA - University of Lyon, FRANCE
E-mail: fabienne.anhalt@ens-lyon.fr
http://perso.ens-lyon.fr/fabienne.anhalt

Johan Montagnat
CNRS - I3S Laboratory, FRANCE
Tel.: +33 4 92 96 51 03
E-mail: johan@i3s.unice.fr
http://www.i3s.unice.fr/~johan

Pascale Vicat-Blanc Primet
INRIA - University of Lyon, FRANCE
Tel.: +33 4 7272 8802
E-mail: pascale.primet@inria.fr
http://perso.ens-lyon.fr/pascale.primet

cost reduction. Four resource allocation strategies relying on the expertise that can be captured in workflow-based applications are considered. Results of these strategies are confined virtual infrastructure descriptions that are interpreted by the HIPerNet engine responsible for allocating, reserving and configuring physical resources. The evaluation of this framework was carried out on the Aladdin/Grid'5000 testbed using a real application from the area of medical image analysis.

Keywords Cloud computing · Resources allocation · IaaS · Workflows · Network virtualization · Description language

1 Introduction

Cloud computing infrastructures are being increasingly exploited for tackling the computation needs of large-scale distributed applications. They provide resources on demand to address the computation needs of the applications. The virtualization technologies exploited ease the migration of heavyweight applications by adapting the execution environment to the specific application requirements. A challenging problem, both for cloud providers and cloud users is the estimation of the amount of resources to allocate out of the cloud for a specific usage. In the commercial cloud offers, various business models have been developed to bill resources usage. They are usually based on a coarse-grained metering of the amount of CPU and disk space consumed. Estimating the proper amount of resources to allocate is left to the responsibility of the user, although such an estimation is far from trivial, especially when considering distributed applications. From a user point of view, assistance in resources consumption planing and cost management is therefore highly desirable. Furthermore, there exist non-commercial platforms for which a finer estimate of the resources allocation process is of interest for the infrastructure providers. Finally, CPU and disk space are not necessarily the only resources that can be provisioned. For instance, network bandwidth is also a critical resource for many distributed applications. This paper addresses the problem of estimating the "optimal" amount of cloud resources (network links and nodes) needed to run complex distributed applications according to various strategies.

From an infrastructure provider point of view, the major challenge is to account (financially or not) for resources usage according to specific criteria (*e.g.* fair share among users, degressive price, etc). Commercial cloud infrastructures use a simple cost computation model (*e.g.* Amazon EC2[1] charges users per hours of resources usage, per GB/month of storage and for the generated trafic in networks) that lets the user responsible for precisely estimating the amount of resources to reserve. This practice is less suitable for dedicated infrastructures, such as academic clouds or intra-enterprise clouds, for which providers are not only interested in billing but also aim at improving quality of services and optimizing resources sharing. Therefore, a finer grain model has to be proposed to (i) decide on the amount of resources to allocate to each application and (ii) compute the resources usage cost.

From a user point of view, the problem of determining the size of the infrastructure to deploy for supporting a given application run is often a difficult one. Although a quasi-unlimited amount of computing resources may be allocated, a trade-off has to be found between (i) the allocated infrastructure cost, (ii) the performance expected and (iii) the optimal performance achievable, that depends on the level of parallelization of the application.

[1] http://aws.amazon.com/ec2/

Without assistance, the user has to resort to a qualitative appreciation of the optimal infrastructure to allocate, based on her previous experience with the application and the cloud computing system used.

Theoretically, the cost of an infrastructure deployment and usage scenario may be quantitatively estimated by the system if sufficient information on the application and the infrastructure is known. In the general case though, it is hardly feasible to anticipate the precise needs of a parallel application or the behavior of such an application given a determined size infrastructure. Restraining the problem a bit more, it appears that workflow-based applications have good properties for such a quantitative estimation. Workflow-based applications represent a large class of coarse-grained distributed applications [16]. Taking advantage of the workflow formalism, the application logic can be interpreted and exploited to produce an execution schedule estimate.

Determining the amount of computational and storage resources needed for each application run is often not sufficient when considering distributed applications. Communication network bandwidth is also a critical resource, shared among the infrastructure users, which may impact application performance significantly. Nowadays, the virtualization paradigm can be applied and combined to both network and computing resources and the Infrastructure as a Service can be extended to the network. This advanced cloud computing paradigm enables the definition of confined execution environments, including the amount of virtual resources needed, virtual network topology and network links bandwidth. The global cloud infrastructure manager is able to create multiple, isolated and protected environments for multiple users concurrently sharing the same set of physical resources without interfering with each others.

The objective of this paper is to develop virtual infrastructures design strategies for cloud computing platforms which size and topology is optimized according to some well-defined metric. These strategies can be used for (i) users to foresee the optimal infrastructure needed for their application given a determined input data set and (ii) infrastructure providers to allocate resources. These strategies are implemented into a framework which allows users to describe and automatically deploy their execution environment. The paper is structured as follows. Section 2 defines the concept of customized virtual private execution infrastructure which extends the Infrastructure as a Service paradigm to the network. Section 3 formulates the workflow-based cost estimation model used to design such execution infrastructures according to four different strategies. Section 4 describes the HIPerNet virtual infrastructure management middleware developed and proposes an experimental validation of our approach using a real distributed application in the area of medical image analysis. Experiments are carried out on the Aladdin/Grid'5000 research infrastructure.

2 Network extension of IaaS paradigm

In the cloud computing context, the network is generally not controlled and network resources are charged according to the total volume of data transferred. Nowadays, networking technology such as network resource virtualization or dynamic bandwidth control enables the flexible provisioning of virtual network resources.

2.1 The VPXI concept

We define the *Virtual Private eXecution Infrastructure* (VPXI) concept as a time-limited interconnection of virtual computing resources through a virtual private overlay network. Any user of a VPXI has the illusion that she is using her own distributed system, while in reality she is using shared cloud resources. The resulting virtual instances are kept isolated from each others. The members of a VPXI have a consistent view of a single private TCP/IP overlay, independently from the underlying physical topology. A VPXI can span multiple networks belonging to disparate administrative domains. Users can join from any location, and deploy and use the same TCP/IP applications they were using on the Internet or their intranet.

A VPXI can be formally represented as a graph in which a vertex is in charge of active data-processing functions and an edge is in charge of moving the data between vertices. A VPXI specification comprises the recursive description of: a) individual computing resources or resource aggregates (clusters) involved, b) performance attributes for each resource element (capacity), c) security attributes, d) commercial attributes, e) temporal attributes, f) elementary functions, which can be attributed to a single resource or a cluster (*e.g.* request of *computing* nodes, *storage* nodes, *visualization* nodes, or *routing* nodes), g) specific services to be provided by the resource (software), h) the virtual-network's topology, including the performance characteristics (typically bandwidth and latency), as well as the security, commercial and temporal attributes of the virtual channels.

Fig. 1 illustrates this concept representing a virtual infrastructure composed by the aggregation of virtual machines interconnected through virtual links. It shows two virtual routers (vertices r_vA and r_vB) which are used to interconnect and perform the bandwidth control among the other virtual resources (vertices r_v 1 to 8). The virtual routers can independently forward the traffic of the different virtual infrastructures which share the same physical network. Each edge represents a virtual link (as l_v1 and l_v2) with different configurations, used to interconnect a pair of virtual resources.

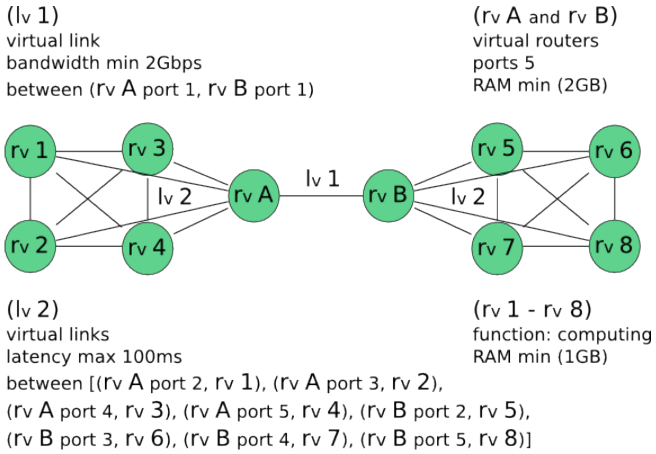


Fig. 1: Example of a VPXI composition using graph notation

2.2 Virtual Infrastructures description

A VPXI is described through the *Virtual eXecution Description Language* (VXDL [24]). VXDL is an XML-based language that allows the user to describe not only the end resources, but also the virtual network's topology, including virtual routers and timeline representation. The VXDL grammar is divided into *Virtual Resources*, *Virtual Network Topology*, and *Virtual Timeline* description as described below. Note that these descriptions are partially optional: it is possible to specify a simple communication infrastructure (a virtual private overlay network) or a simple aggregate of end resources without any network topology description (a virtual cluster or grid).

Virtual Resources Description. This part of VXDL grammar enables users and applications to describe, in a simple and abstract way, all the required end hosts and host groups. VXDL allows the basic resource parametrization (*e.g.* minimum and maximum acceptable values for RAM memory and CPU frequency). An important feature of VXDL is that it proposes cross-layer parameters. With the specification of *anchor* and *the number of virtual machines allocated per physical host* users can directly interact with lower layers and transmit application-specific information. The *anchor* parameters corresponds to a physical allocation constraint of a VPXI. Indeed, in theory a VPXI can be allocated anywhere in a virtualized substrate, but sometimes it is desirable that a virtual end host (or group) be positioned in a given physical location (*e.g.* a site or a machine - URL, IP) for an application-specific reason. On the other hand, in a virtualized substrate, multiple virtual machines can be allocated in the same physical host, sharing the real resources. VXDL enables the definition of a maximum number of virtual machines that must be allocated in a physical host, enabling users to interact directly with the allocation algorithm.

Virtual Network Topology Description. VXDL brings two original aspects within the network's topology description: (i) the joined specification of network elements and computing elements and (ii) the link-organization concept, which permits a simple and abstract description of complex structures. Links can define connections between end hosts, between end hosts and groups, inside groups, between groups and VXrouters, and between VXrouters. In VXDL grammar, the definition of *source - destination pairs* for each link is proposed. The same link definition can be applied to different pairs, simplifying the specification of complex infrastructures. For example, links used to interconnect all components of an homogeneous group, as a cluster, can all be defined in a same link description. Each link can be defined by attributes such as latency, bandwidth, and direction. Latency and bandwidth can be defined by the maximum and minimum values.

Virtual Timeline Description. Any VPXI can be permanent, semi-permanent, or temporary. The VPXI are allocated for a defined lifetime in time slots. Time slots duration is specific to the substrate-management framework and consequently this parameter is configured by the manager of the environment. Often the VPXI components are not used simultaneously or all along the VPXI lifetime. Thus, the specification of an internal timeline for each VPXI can help optimizing the allocation, scheduling, and provisioning processes. Periods can be delimited by temporal marks. A period can be activated after the end of another period.

3 Virtual infrastructure design optimization problem

The virtual infrastructure design optimization problem is to determine, given a specific application run to carry out, both the amount of resources and the network topology needed. Minimizing the amount of resources needed is not necessarily the best objective function. Indeed, it may be better to find a good trade-off between an execution infrastructure cost and the application performance. Such an objective can be formulated as a cost function whose parameters depend on the allocated infrastructure size. Both computing resources and network bandwidth have to be considered in this cost function.

The general optimization problem is intractable given that it depends on the exact distributed application execution behavior. However, restraining the problem to workflow-based applications makes it possible to exploit the knowledge on the application captured by the formal workflow description to address this problem. Many coarse-grained distributed application can be modeled as workflows of service invocation sequences. The workflow directed graph features the application services to be executed (workflow nodes) and the dependencies between these services (edges). As will be discussed later, only acyclic workflows for which the execution schedule can be statically determined are considered in this paper. An example application workflow, used later on in the experimental validation, is shown in Fig. 11. In this case, there are six services which are interconnected by data dependencies. The workflow describes the application computational logic independently from the actual data sets to be processed. Many workflow engines have been proposed to scale the execution for a specific input data set [37]. Each application service might be invoked a variable number of times depending on the data set size and, as long as no dependency exists between two of these invocations, they can be performed concurrently to exploit distributed resources.

3.1 Cost model for workflow-based applications

In our approach, an execution can occur in several *stages*. For each stage, the VPXI can be reallocated with respect to a specific configuration, to perform the execution of part of the workflow. After completing the execution, allocated resources are returned to the cloud. The VPXI reconfiguration between different stages, which may involve redeployment of resources, is time-consuming. One extreme condition, is to create a static VPXI for the whole duration of the complete workflow execution, thus sparing the redeployment cost. Another extreme, is to allocate new resources one by one on demand. The cost model proposed below makes a fine-grained estimate of the resources that will be consumed for each application run. Note that this model is applicable to estimate the cost of a single run of an application on the infrastructure. It does not take into account other costs, such as the long term storage of data onto the cloud storage service. Should users need data storage before and/or after execution, they would be charged additionally and independently of the cost calculated below.

All parameters used in the cost model described below are summarized in table 1. Let $m_{\max}$ be the maximum number of computing nodes available on the infrastructure and s be the number of execution stages of the application. The vector $m = (m_1, m_2, ..., m_s)$ is the number of nodes used at each execution stage with $\forall i, m_i \leq m_{\max}$. Let c_r be the per-

Table 1: Notations used in the cost function model.

$m_{\max}$	maximum number of computing nodes available on the infrastructure
n	number of input data items
s	number of execution stages of the application
$m = (m_1, m_2, ..., m_s)$	number of nodes used at each execution stage with $\forall i, m_i \leq m_{\max}$
c_r	per-second cost of a computing resource
c_b	per-Mbps cost of bandwidth
Td_i	deployment time of stage i (in seconds)
$T_i(m_i, n, b)$	execution time of stage i (in seconds)
$b = (b_1, b_2, ..., b_{ki}), i \in [1..s]$	links bandwidth used at stage i (in Mbps)

second cost of a computing resource. The total computing cost of the infrastructure allocated for the application is:

$$C_r = c_r \times \sum_{i=1}^{s} m_i \times (Td_i + T_i(m_i, n, b)) \tag{1}$$

where Td_i is the deployment time (including resource reservation and initialization time) and $T_i(m_i, n, b)$ is the execution time at stage i. T_i depends both on computing time and data transfer time involved within stage i. It is parameterized by the number of resources reserved (m_i), the number of input data items to process (n) and the bandwidth ($b = (b_1, b_2, ..., b_{ki}), i \in [1..s]$) of the network links used for data transfer. The computation of T_i is possible using the application logic described through the workflow. The workflow engine used in our experiment, MOTEUR [16], was seminally designed to produce an execution schedule and control the distribution of an application at runtime. It was enriched with a resource allocation and scheduling planner that is used to estimate T_i, given that information on the workflow services execution time and transferred data amount is available.

The total infrastructure cost is also impacted by the data transfer time. If the per-Mbps cost of the reserved bandwidth is c_b, then the total data transfer cost is:

$$C_b = c_b \times \sum_{i=1}^{s} (Td_i + T_i(m_i, n, b)) \sum_{j=1}^{k_i} b_j \tag{2}$$

This cost applies to an infrastructure where the amount of network bandwidth allocated is controlled (*e.g.* HIPerNet [23]). It sums all data transfer costs involved in the workflow execution, including workflow input data transferred from outside the cloud (at stage 1), the temporary data generated during workflow execution (at all stages) and the output data transferred to external resources (at stage s).

From formulas 1 and 2, the total infrastructure cost to execute the application can be computed:

$$C = C_r + C_b \tag{3}$$

This cost has to be optimized considering a maximum admissible cost and the application performance scalability. A trade-off has to be found between the amount of computing resources and network resources allocated (which impacts T_i), and the resulting cost.

3.2 Comparison to a commercial offer

The cost model described in equations 1 and 2 can be used for cost estimation both from an infrastructure provider and an infrastructure user point of view. Depending on the intended usage, it may be tuned. For instance, Amazon EC2 cloud computing offer charge users per hour, day or week of usage. The times estimated are therefore rounded at the ceil value in the unit considered. In addition, Amazon EC2 does not account for infrastructure deployment time in billing ($Td_i = 0$). This cloud infrastructure also does not make it possible to adapt nor guarantee the network bandwidth allocated. The amount of network resources is therefore billed on the basis of the total amount of data transferred rather than the amount of bandwidth consumed. Finally, Amazon charges for workflow input and output data transfers (data transfer from and to the storage resources outside the cloud) additionally, while in the model proposed above this transfer is accounted for in C_b (equation 2).

Consequently, the cost billed for the EC2 computing resources usage is one of:

$$C'_r = \begin{cases} c'_r \times m_{\max} \times \left\lceil \dfrac{\sum_{i=1}^{s} T_i(m_i, n, b)}{3600} \right\rceil & \text{(4a)} \\ c'_r \times \sum_{i=1}^{s} m_i \times \left\lceil \dfrac{T_i(m_i, n, b)}{3600} \right\rceil & \text{(4b)} \end{cases}$$

where c'_r is the Amazon EC2 per-hour unit cost of computing resources. Case 4a applies if a single reservation is made for the whole duration of the workflow execution. In that case, there is a single stage and the maximum number of resources ($m_{\max}$) will be reserved. Case 4b applies if one reservation is made for each stage. Compared to equation 1, the cost computed in equation 4 is impacted by rounding to the next hour. In particular in case of multiple reservations (case 4b), the rounding at each stage may be penalizing. A trade-off has to be found between reserving the maximum number of resources for the whole duration of the computation (case 4a) and adapting the number of resources at each stage, at the expense of an over-estimated platform usage time (case 4b).

Similarly, the cost charged for usage of network resources when transferring input/output data in Amazon EC2 is:

$$C'_b = c'_b \times V_D \tag{5}$$

where V_D is the total amount of data transferred between EC2 and other data sources (*e.g.* the user machine or a database server on Amazon S3), and c'_b is a per-volume unit cost. Unlike equation 2, this cost cannot be adapted to specific network usage requirements. This reflects the fact that this infrastructure does not provide any bandwidth control mechanism.

The total Amazon EC2 cost is:

$$C' = C'_r + C'_b \tag{6}$$

3.3 VPXI design strategies

The application execution time for each stage (T_i) depends on the amount of resources allocated within each VPXI. Four strategies are described below to determine VPXIs and estimate the corresponding execution times.

3.3.1 Naive strategy

Given p the number of services composing an application workflow and t_i the benchmarked execution time of service $i \in 1..p$, a set of $m_{\max}$ virtual computing nodes is allocated and split proportionally to each service execution time: $m_{\max} t_i / \sum_j t_j$ nodes are dedicated to the service i. The network bandwidth is similarly allocated proportionally to the amount of data to transfer between each pair of services, or the same bandwidth is reserved for all links in the infrastructure. This strategy is naive in the sense that it only considers a single execution stage and the resources are statically allocated to each service even though a service may not be involved during the whole duration of the workflow execution. This strategy serves as a performance base-line.

3.3.2 FIFO strategy

In this approach, we make the simplifying assumption that all services can be deployed on every computing resources. These resources are thus indistinguishable and the scheduler may request any task to be executed on any resource. A *FIFO* scheduling strategy is optimal in this case and a single stage is considered since infrastructure redeployment is unnecessary ($T = T_1$). In addition, the same bandwidth is reserved for all links in the infrastructure ($b_1 = b_2 = ... = b_k$). As an example, Fig. 2 displays the estimated execution time and the total cost of the workflow from Fig. 11 with regard to the bandwidth (for $n = 32$ input data items and unit costs $c_r = c_b = 0.2$). When the bandwidth is small, the total cost is high due to the data transfer time. When the bandwidth increases, the execution time and cost both decrease. However, after a 2.0Mbps threshold, the execution time only slightly reduces while the bandwidth allocation cost increase dominates. The optimization method used to numerically approximate the optimal bandwidth leads to 0.6517Mbps.

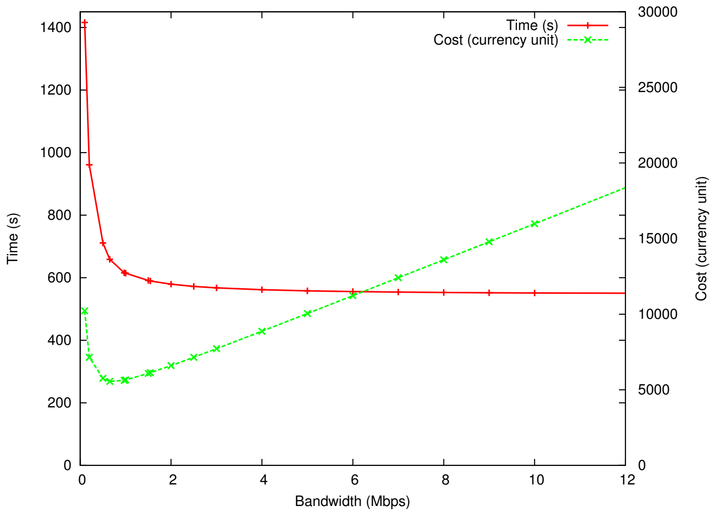


Fig. 2: Estimation of the execution time and total cost with regard to the bandwidth of the *FIFO* strategy

3.3.3 Optimized strategy

The *FIFO* strategy can only apply with identical resources and without optimizing the bandwidths between each pair of resources. Conversely, the optimized strategy described below considers dividing the workflow execution in multiple stages and allocating resources and bandwidth independently for each stage. The cost minimization algorithm is executed for each stage to allocate an optimal number of virtual resources to the services involved in this stage.

An algorithm is needed to decide on the number of stages and when infrastructure reconfiguration should happen. Firstly, the workflow of services is transformed into a *Directed Acyclic execution Graph* (DAG), using the second composition approach presented in [39] for instance. Secondly, the DAG is divided in execution stages, each of them meant to be executed on a specific virtual infrastructure. An example execution DAG for the workflow of Fig. 11 is shown in Fig. 3, where `IN` and `OUT` are special entry and exit nodes that are not accounted for in the execution and data transfer times estimation. The pseudo-code of the DAG split into stages is presented in algorithm 1. An execution stage is defined as the set of invocations which have the same depth in the DAG graph.

Note that the DAG generation is only possible for workflows without unbounded loops (the exact number of invocations of each service needs to be known) so that the workflow planer can determine a complete execution schedule. Workflows including *while* kind of loops, or *foreach* constructs iterating over unknown size data structures make the workflow unresolvable prior to execution. This is limiting the class of applications that can be planed. Yet, this represents a broad category of workflow applications in e-Science (many data-intensive, scientific workflow languages do not support loops [12]). A solution for dealing with workflows with unresolvable constructs is to divide them into smaller resolvable sub-workflows. This generation process has to be revised dynamically though (*e.g.* each time a loop is iterated, the loop body sub-workflow can be generated). Such a strategy was implemented in the workflow manager of the DIET middleware (MA DAG) for instance, to deal with workflows which could not be represented by DAGs[2].

Algorithm 1 Execution DAG split into stages

```
Require: processedServices list initialized with all workflow inputs.
Require: stage = 1

while There are still services to process do
    stage-services = empty list
    for each service S in workflow do
        if all inputs of S come from the list of processed services then
            add S into stage-services
            set stage of service S to stage
        end if
    end for
    add list stage-services to list processedServices
    increment the stage counter (stage = stage + 1)
end while
```

At each execution stage, the infrastructure is reconfigured for only deploying the specific services involved in that stage. The resources are allocated proportionally to the number of

[2] DIET MA DAG: http://graal.ens-lyon.fr/~diet/workflow.html

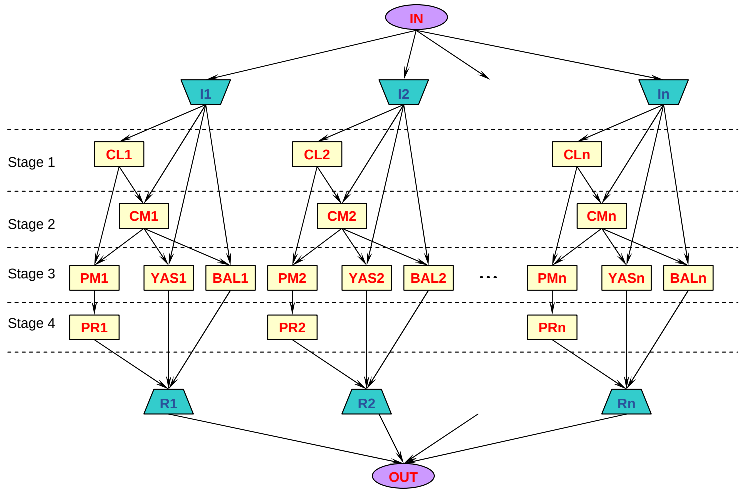


Fig. 3: DAG jobs of Bronze Standard application for n inputs

invocations needed for each service. In a typical data intensive application execution, there are more data items to process (n) than resources available ($m_{\max}$). For instance, in the case of a stage i with only one service S (*e.g.* stage 1, 2 or 4 in Fig. 3), $m_{\max}$ data items are processed concurrently by S and the process is repeated $n/m_{\max}$ times, leading to the execution time:

$$T_i = \left\lceil \frac{n}{m_{\max}} \right\rceil \times T_S \tag{7}$$

where T_S is the execution time for S.

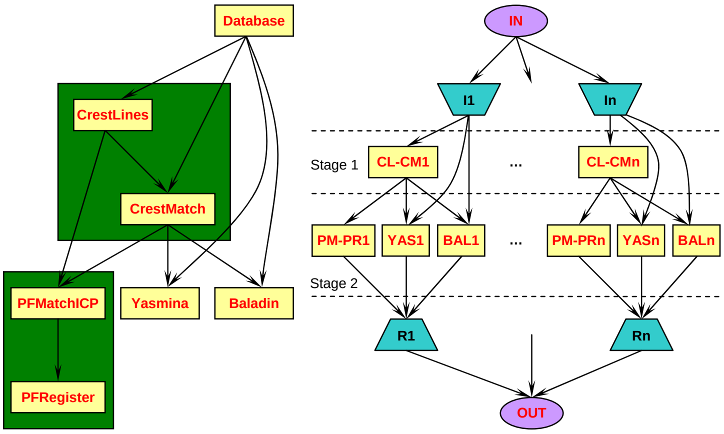
More generally, the optimal resources and bandwidth allocation strategy, taking into account the number of service invocations, the execution time and the data transfer time in each stage is computed using the multi-criterions *Downhill Simplex* minimization method. Let $inv_j, j = 1..s$ be the number of invocations of service j at stage i where s is the number of services being executed at this stage. Let vector $m = (m_1, m_2, ..., m_s)$ be a combination of number of resources allocated to the service j. This combination must satisfy the condition $\sum_{j=1}^{s} m_j \leq m_{\max}$. The resulting optimal execution time to complete inv_j invocations of service j is:

$$T_j = \left\lceil \frac{inv_j}{m_j} \right\rceil \times T_{uj} \tag{8}$$

where T_{uj} is the unit execution time of service j.

3.3.4 Services grouping optimization

The total execution cost also depends on the infrastructure deployment time of each stage. An optimization of the total resources reservation and redeployment time was designed, extending the job grouping strategy without loss of parallelism introduced in [15]. This strategy minimizes the application makespan by grouping services which would have been executed sequentially, thus reducing data transfers and the number of job invocations needed. Applying this strategy to the workflow of Fig. 11, two services groups are identified which do not



(a) Grouping without parallelism loss (b) Resulting execution DAG considering n input data items

Fig. 4: Services grouping without parallelism loss

cause loss of parallelism as shown in Fig. 4a. The number of execution stages can also be reduced as shown in Fig. 4b.

This strategy only exploits workflow topology information but not the actual execution cost of the services, although it might be preferable to loose some degree of parallelism, when the grouping gain is higher. The trade-off can be found thanks to the execution planner developed for the allocation strategies. Starting from the execution DAG split into stages, job invocation groups are evaluated for each consecutive pair of stages. For each service A of the workflow involved in the stage i, let $B_0, B_1, ..., B_j$ be all children from A in stage $i+1$. All possible combinations of grouping A with one or more of the B_k services is tested and the resulting execution cost is evaluated by optimizing the number of resources and the bandwidth allocated. In the example used throughout this paper, the best solution is shown in Fig. 5.

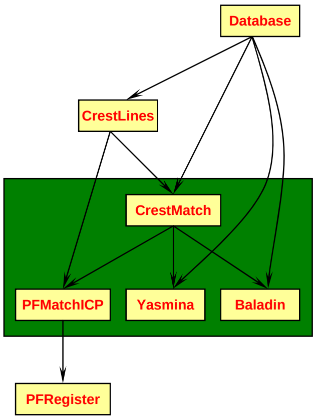


Fig. 5: Grouping CrestMatch, PFMatchICP, Yasmina and Baladin

```
<vxdl:resource>
  <vxdl:id>database </vxdl:id>
  <vxdl:ramMemory>
    <vxdl:min>1</vxdl:min>
    <vxdl:minUnit>GB</vxdl:minUnit>
   </vxdl:ramMemory>
 </vxdl:resource>
```

Fig. 6: Generic part description in VXDL language

```
<vxdl:group>
  <vxdl:id>Cluster_Service_i </vxdl:id>
  <vxdl:function>
    <vxdl:id>computing </vxdl:id>
   </vxdl:function>
  <vxdl:size>
    <vxdl:min>m_i </vxdl:min>
   </vxdl:size>
  <vxdl:resource>
    <vxdl:id>Node_Cluster_Service_i </vxdl:id>
    <vxdl:ramMemory>
      <vxdl:min>512</vxdl:min>
      <vxdl:minUnit>MB</vxdl:minUnit>
     </vxdl:ramMemory>
   </vxdl:resource>
 </vxdl:group>
```

Fig. 7: VXDL description of one cluster of computing resources

3.4 Virtual resources description generation

For each VPXI design strategy, a VPXI is described through VXDL. This VPXI is composed of two parts: a *generic* part and *variable* one. The *generic* part is used to describe mandatory nodes to execute an application (*e.g.* middleware, database). In our model, we use one node for this part. It is used for the database server storing the input data, intermediate and final results. Fig. 6 presents the description of this part. The *variable* part composing of computing resources is generated according to the design strategies presented in section 3.3.

The *naive* strategy divides the set of m virtual computing resources proportionally to the execution time of workflow services. We use the `<vxdl:group>` tag to describe a cluster of virtual computing resources corresponding to a workflow service. This cluster composes of $m_i = m_{\max} t_i / \sum_j t_j$ resources with a minimum amount of RAM. Fig. 7 presents the description of this strategy.

Similarly to the *naive* strategy, the *FIFO* strategy runs the application in a single stage and assumes that all services can be deployed on every computing resources. Therefore, the VXDL description has only one group.

The *optimized* strategy has a more complex description which uses the *Virtual Timeline Description* of VXDL language. Fig. 8 presents an example of the application which has two stages. The first stage has one service which executes in t_i seconds. The second stage has three services starting at the same time after first stage has finished.

```
<vxdl:virtualTimeline>
   <vxdl:id>ApplicationTimeline </vxdl:id>
   <vxdl:timeline>
       <vxdl:id>T1</vxdl:id>
       <vxdl:activate>Service_i </vxdl:activate>
       <vxdl:until>
           <vxdl:totalTime>ti </vxdl:totalTime>
           <vxdl:totalTimeUnit>s</vxdl:totalTimeUnit>
       </vxdl:until>
   </vxdl:timeline>
   <vxdl:timeline>
       <vxdl:id>T2</vxdl:id>
       <vxdl:after>T1</vxdl:after>
       <vxdl:activate>Service_j1 </vxdl:activate>
       <vxdl:activate>Service_j2 </vxdl:activate>
       <vxdl:activate>Service_j3 </vxdl:activate>
       <vxdl:until>
            <vxdl:totalTime>tj </vxdl:totalTime>
            <vxdl:totalTimeUnit>s</vxdl:totalTimeUnit>
       </vxdl:until>
   </vxdl:timeline>
</vxdl:virtualTimeline>
```

Fig. 8: Virtual Timeline Description for the *optimized* strategy

```
<vxdl:virtualTopology>
   <vxdl:id>VirtualNetwork </vxdl:id>
   <vxdl:link>
    <vxdl:id>lv 1</vxdl:id>
        <vxdl:bandwidth>
             <vxdl:min>2</vxdl:min>
             <vxdl:minUnit>Mbps</vxdl:minUnit>
        </vxdl:bandwidth>
        <vxdl:direction>bi </vxdl:direction>
        <vxdl:pair>
            <vxdl:source>database </vxdl:source>
            <vxdl:destination>cluster_service_i </vxdl:destination>
       </vxdl:pair>
   </vxdl:link>
</vxdl:virtualTopology>
```

Fig. 9: Virtual network topology description

The virtual network topology is specified by depending on each application. The more dependence between workflow services, the more complicated network topology. Each link is specified by a minimum amount of bandwidth and one or more pairs of source/destination. Fig. 9 shows a typical link between the database storing the workflow input and the computing resource cluster of a workflow service.

4 Validation on the Aladdin/Grid'5000 testbed

The cloud nodes and network allocation framework described in this paper is implemented using the HIPerNet middleware [3,35], designed in the context of the HIPCAL project[3]. HIPerNet manages a set of VPXIs as illustrated in Fig. 10 where two virtual execution infrastructures (*VPXI A* and *VPXI B*) are represented. Each application can execute, confined in a VPXI dedicated for a defined time period. Our workflow application manager was instrumented with the HIPerNet API to make it able to control the cloud resources allocation.

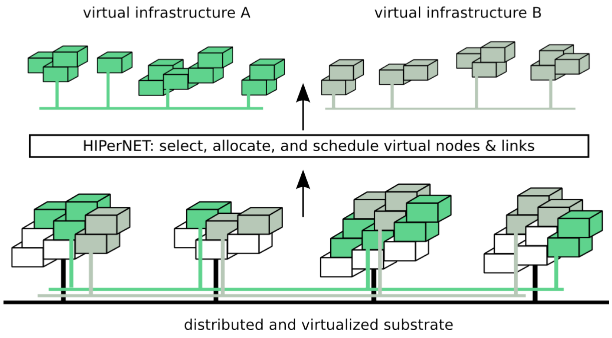

virtual infrastructure A
virtual infrastructure B
HIPerNET: select, allocate, and schedule virtual nodes & links
distributed and virtualized substrate

Fig. 10: Example of a VPXI allocation on a distributed and virtualized HIPerSpace.

4.1 HIPerNet framework and Grid'5000 substrate

HIPerNet provides a framework to build and manage private, dynamic, predictable, and large-scale virtual computing environments, that high-end challenging applications can use with traditional APIs: standard POSIX calls, sockets, and Message Passing (*e.g.* MPI and OpenMP) communication libraries. With this framework, a user preempts and, for a given timeframe, virtually interconnects a pool of virtual resources from a distributed physical substrate, in order to execute her application. VPXIs correspond to the HIPerNet's management unit.

The HIPerNet framework aims at partitioning a distributed physical infrastructure (computers, disks, and networks) into dedicated virtual private computing environments composed dynamically. When a new machine joins the physical resource set, HIPerNet prepares its operating system to enable several virtual machines (VMs) to be instantiated dynamically when required. This set of potential virtual machines is called an HIPerSpace and it is represented in the HIPerSpace database. The HIPerSpace is the only entity that sees the physical entities. A resource, volunteer to join the resource pool, is automatically initiated and registered in the HIPerSpace database. The discovery of all the devices of the physical node is also automatic. An image of the specific HIPerNet operating system is deployed on it. In our current HIPerNet implementation, the operating system image basically contains the Xen Hypervisor and its domain of administration called domain 0 (Dom 0). The HIPerSpace registrar (operational HIPerVisor) collects and stores data persistently, and manages accounts (*e.g.* the authentication database). It is therefore hosted by a physical machine

[3] http://www.ens-lyon.fr/LIP/RESO/Projects/HIPCAL/ProjetsHIPCAL.html

outside of the HIPerSpace itself. For the sake of robustness and scalability, the HIPerSpace registrar can be replicated or even distributed.

When a user submits a VPXI request specified using the VXDL language, the HIPerNet allocator examines the request and executes an embedding algorithm to map the virtual infrastructure on the physical one. Then if the request is accepted, HIPerNet deploys or reconfigures the virtual resources of the VPXI according to this specification. Using the bandwidth control concept in Grid'5000 [34], HIPerNet allocates link bandwidth to all the virtual links whose bandwidth was explicitly specified in VXDL during the request submission. When a VPXI is created, virtual links are provisioned according to the VXDL request. Within this request, the user can specify several stages for the VPXI, involving different configurations of bandwidth to best fit the application's requirements. While the VPXI is running, the user can change its configuration moving from one stage to another.

The experiments are carried out using several virtual infrastructures managed by HIPerNet within the Aladdin/Grid'5000 testbed[4]. Aladdin/Grid'5000 enables a user to request, reconfigure, and access physical machines belonging to 9 sites distributed in France. In our experiments, several Aladdin/Grid'5000 nodes were reserved to compose a pool of physical resources that we initialize to form a HIPerSpace. To instantiate an HIPerSpace, specific tools provided by the testbed are used. This is the only part aware of the physical infrastructure of the HIPerNet middleware. All the other parts are independent of the physical resources because they use them indirectly through the services provided by HIPerNet.

4.2 Test application

The experiments are performed using the *Bronze Standard* (BS) a real workflow-based application from the area of medical image analysis [17]. The BS technique tackles the difficult problem of validating medical-image analysis tools. As there is usually no reference, or gold standard, to validate the result of a medical image analysis algorithm, it is very difficult to objectively assess the results' quality. The BS technique statistically quantifies the maximal error resulting from widely used image registration algorithms. The larger the sample image database and the number of registration algorithms to compare with, the most accurate the method. This procedure is very scalable and described through a complex application workflow illustrated in Fig. 11. In the experiments reported below, a clinical database with 59 pairs of patient images was used. For each run, 354 computing tasks were generated.

4.3 Experiments

For testing the allocation strategies, a system image containing the OS (based on a Debian *Etch* Linux distribution with a kernel version 2.6.18-8), the domain-specific image processing services was created. The infrastructures allocated are managed by the HIPerNet framework which enables the joint virtualization of computing and network resources. The physical resources were reserved on the fully reconfigurable Aladdin/Grid'5000 research infrastructure, cluster *sagittaire* in Lyon, France. The physical resources are Sun Fire V20z machines, 2.4GHz, 2 cores and 2GB RAM interconnected through 1Gbps Ethernet. The experimental infrastructure is diagrammed in Fig. 12. For all experiments, 36 physical computers were reserved. The MOTEUR workflow engine, as a client of the HIPerNet cloud

[4] https://www.grid5000.fr

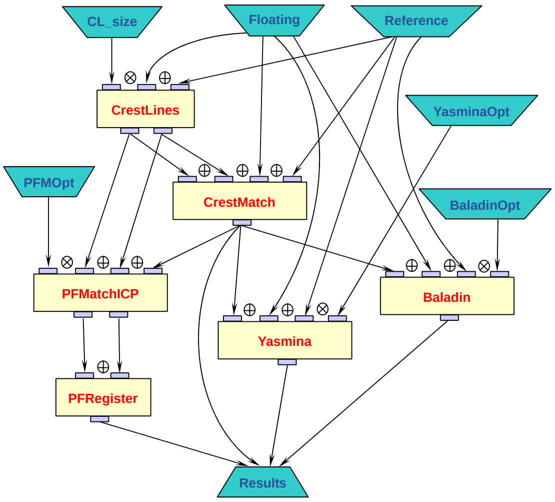


Fig. 11: Bronze Standard workflow.

manager engine, was hosted on one physical host, outside of the cloud. The 35 remaining computers were registered in the HIPerSpace. The HIPerNet engine deploys and manages virtual machines on these computer on demand (dark arrows), either with on OS image of the input database server or the application services. In our experiments, each physical computer hosts a single virtual machine. MOTEUR produces VXDL descriptions that are requested to the HIPerNet engine (blue connection). After receiving all virtual machines allocated to the VPXI, MOTEUR connects to the computing nodes to invoke the application services (red connections). The computing nodes connect to the database host to copy the input data and send the computational results, and the final results are sent to MOTEUR (green connections).

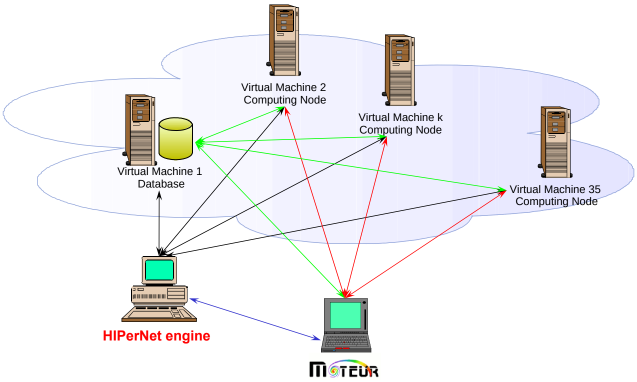


Fig. 12: Experimental infrastructure

Table 2: Benchmark of the BS services execution time and data transfer volumes.

Services	Time (average ± standard deviation)	Input data	Produced data
CrestLines	31.06s ± 0.57	15MB	10MB
CrestMatch	3.22s ± 0.51	25MB	4MB
PFMatchICP	10.14s ± 2.41	10.2MB	240kB
PFRegister	0.64s ± 0.22	240kB	160kB
Yasmina	52.94s ± 12.96	15.2MB	4MB
Baladin	226.18s ± 19.36	15.2MB	4MB

For the needs of the MOTEUR planner, all 6 services involved in the BS workflow have been benchmarked for execution time and amount of data transferred as reported in table 2. For each experiment, the application was executed 5 times and the makespan was averaged to minimize the execution time variations encountered in distributed computing. The standard deviation is also reported.

For each strategy, the planner optimizer was executed to determine the configuration with the minimal execution cost. The number of virtual machines allocated to the application and the bandwidth between the database node and computing nodes are specified by corresponding VXDL documents.

4.3.1 Single stage strategies

The *naive* and *FIFO* strategies are single-stage. They use maximum available computing resources (34 computing machines) with an optimal bandwidth yielding to a minimal execution cost. The virtual infrastructures of the *naive* and *FIFO* strategies are represented in Fig. 13 and 14, respectively. Conversely, the *optimized* strategies are multi-stages, optimize bandwidth needed, and may allocate less resources than the maximum available when there is no gain in doing so.

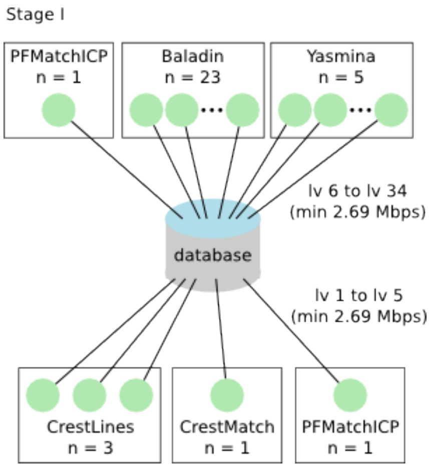


Fig. 13: Virtual Infrastructure composition considering *naive* strategy.

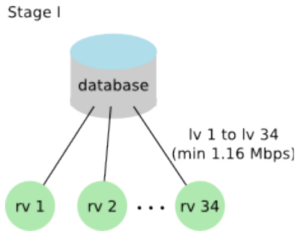


Fig. 14: Virtual Infrastructure composition considering *FIFO* strategy.

We also measured the deployment time of the virtual infrastructure before running the application and the reconfiguration time between stages of the *optimized* strategies. The reconfiguration time takes into account bandwidth reconfiguration between the database host and computing nodes allocated to application services in each stage. The virtual machines in stage n are reused in stage $n + 1$. If the stage $n + 1$ use more virtual machines than stage n, additional virtual machines are deployed during the execution of stage n.

The *naive* allocation strategy allocated the 34 computing nodes to application services as follows: 3 nodes for CrestLines, 1 node for CrestMatch, 1 node for PFMatchICP, 1 node for PFRegister, 5 nodes for Yasmina, and 23 nodes for Baladin. The same bandwidth, 2.69Mbps, is used for all computing nodes. The application makespan is 67.08min ± 0.10min. This experiment shows that the virtual resources are not well exploited during the execution. Fig. 15 shows a schedule of this strategy. Each colored line represent one task duration: it starts once the corresponding task has been submitted and stops at the end of its execution. The first, brighter part of the line represents the task waiting time spent from submission until a resource becomes available for execution. Colors are arbitrary and just help to distinguish the different tasks. As can be seen, at the beginning of the execution, only three nodes are used to execute the CrestLines service. Other resources are wasted. Similarly, the result of CrestMatch is needed for three services: PFMatchICP, Yasmina and Baladin but there is only one resource allocated to this service according to this strategy and it becomes a bottleneck.

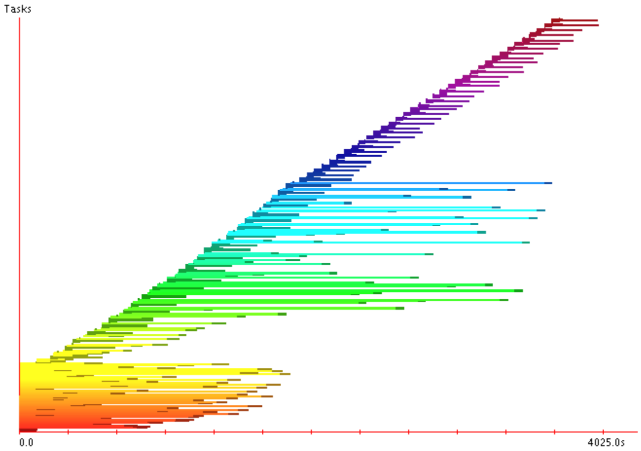


Fig. 15: Tasks schedule with the *naive* strategy

The makespan of the *FIFO* strategy is lower: 46.88min ± 0.78min with the optimal bandwidth (1.16Mbps). The standard deviation of this strategy is higher due to the variable arriving order of the tasks. Some long tasks can be executed on the same computing resource, leading to the increase of the application makespan. Fig. 16 shows a typical task schedule for this strategy.

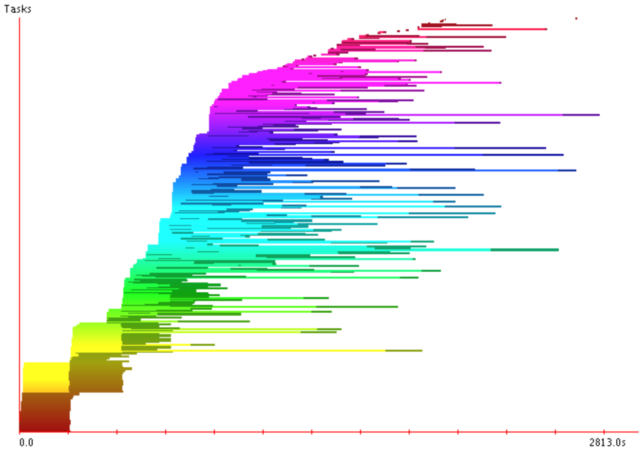


Fig. 16: Tasks schedule with the *FIFO* strategy

4.3.2 Multi-stages strategies

For the *optimized* strategies, the planer determines the number of virtual resources and the bandwidths yielding to a minimal execution cost. Without services grouping there are 4 execution stages which are represented in Fig. 17. According to the optimization results: only 30 nodes were allocated for the first, second and fourth stages (additional resources would be wasted). The bandwidths are 4.62Mbps, 14.74Mbps and 3.87Mbps, respectively. For the third stage, 4 nodes were allocated to PFMatchICP, 6 nodes for Yasmina and 20 nodes for Baladin. The bandwidth for each service in this stage is 0.87Mbps, 1.36Mbps and 1.29Mbps, respectively. The corresponding application makespan is 37.05min ± 0.25min.

Further grouping the application services as shown in Fig. 5, the application is divided into three stages only, using 30 nodes each. As presented in Fig. 18, the bandwidth allocated for each stage is 4.90Mbps, 1.95Mbps and 3.87Mbps, respectively. The application makespan is then 22.93min ± 0.35min. Besides the execution time improvement, the number of resources consumed is also lowered. As we can observe in Fig. 19, all tasks of the same stage do not finish exactly at the same time though, due to some variations of the image analysis tools execution time depending on the exact processed image content. This has an impact as the tasks of stage n have to wait for the longest task of stage $n-1$ before the system can be reconfigured.

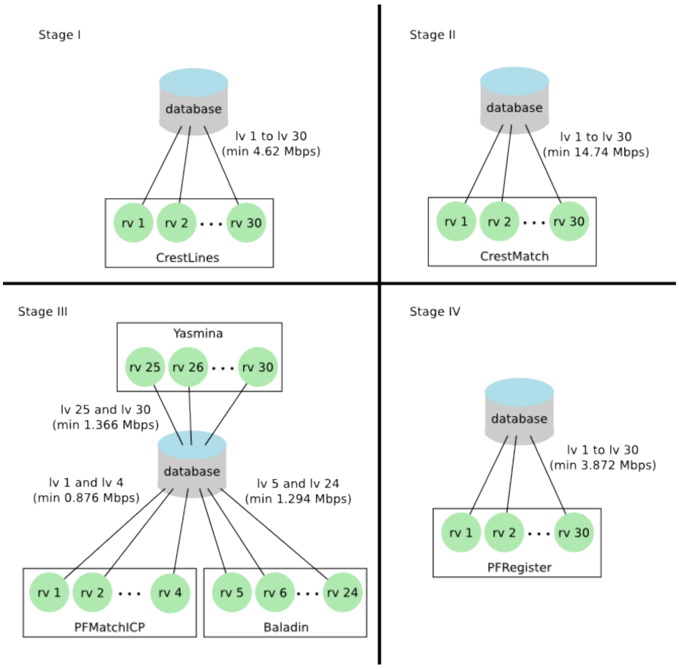


Fig. 17: Virtual Infrastructure composition considering *optimized* strategy without grouping services

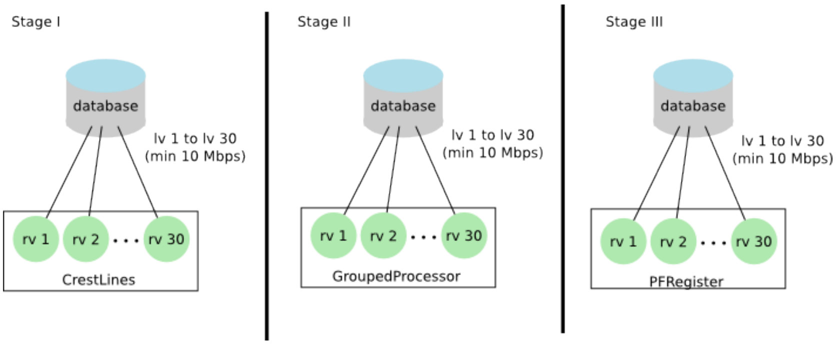


Fig. 18: Virtual Infrastructure composition considering *optimized* strategy with grouping services

4.3.3 Summary

In conclusion, table 3 compares the performance of the strategies presented above and the associated platform cost computed using equation 3. The worst case is the *naive* strategy that uses the maximum number of resources for a very large makespan and a long deployment. The *FIFO* strategy spends the same time to deploy the infrastructure but it has a better makespan than the *naive* strategy. The *naive* and *FIFO* strategies reconfiguration time is null since they are single-stage. The *optimized* strategy without grouping services has better

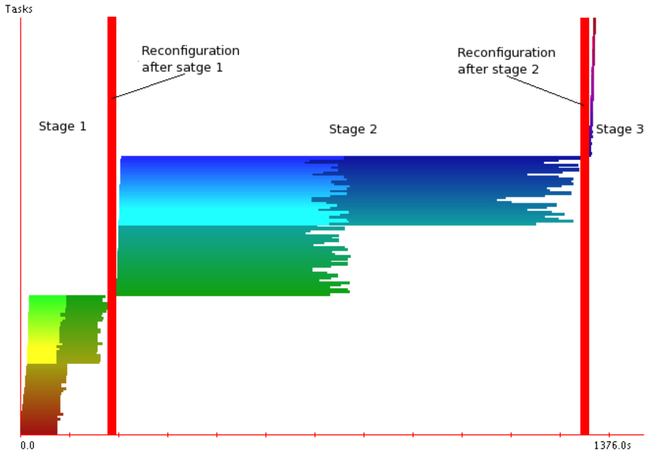


Fig. 19: Tasks schedule with *optimized* services grouping

results both in terms of application makespan and number of resources consumed than the *naive* and *FIFO* strategies, although it has to spend time to reconfigure the infrastructure after each stage. The best case is obtained for the *optimized* strategy with services grouping. It uses less resources, spends less time to reconfigure the infrastructure and returns the results faster. In terms of the deployment time, the *naive* and *FIFO* strategies take 29.83min to deploy 35 virtual machines. It is to be noted that HIPerNet does not enable the parallel deployment of resources yet. This duration corresponds to the time needed to copy the OS images (319MB) from the HIPerNet engine to the virtual machines and start them sequentially. The *optimized* strategies use only 31 machines, reducing the deployment time to 25.68min. In the future, parallel deployment is expected to lower this redeployment overhead. As expected, the cost estimated is lowered for higher performing strategies to the reduction of the application makespan and of the network bandwidth consumed.

Table 3: Performance comparison between the four strategies

Strategy	Makespan	#VM	Deployment time	Reconfiguration time	Execution cost $(\times 10^5)$
Naive	67.08min ± 0.10	35	29.83min	0	$1.40 \times c_r + 3.68 \times c_b$
FIFO	46.88min ± 0.78	35	29.83min	0	$0.98 \times c_r + 1.10 \times c_b$
Optimized (without grouping)	37.05min ± 0.25	31	25.68min	79.29s	$0.69 \times c_r + 0.98 \times c_b$
Optimized (with grouping)	22.93min ± 0.58	31	25.68min	52.86s	$0.42 \times c_r + 0.48 \times c_b$

4.3.4 Comparison with a commercial offer

Table 4 presents the cost billed by Amazon EC2 (equation 6) as a function of the unit costs (currently in Europe, c'_r =$0.10 / VM / hour, and c'_b =$0.15 / day / GB). For these computations we made the hypothesis of the same running times on Amazon EC2 nodes as on the Aladdin/Grid'5000 platform. While Amazon EC2 data transfer cost is the same for all strategies (transfer of 1GB input and output data), the cost payed for computing resources varies. The *naive* strategy, executing in more than one hour, dominates the reservation cost for computing resources ($2 \text{ hours} \times 35\text{VMs} \times c'_r$). The reservation duration reduces to one hour for other strategies. Since the *FIFO* strategy uses 35VMs, its cost is higher than the *optimized* strategy with and without grouping optimization which use less resources (31VMs). Compared to Amazon EC2 cost, the cost model introduced in this paper is not rounded to the next hour, thus showing a decrease of the execution cost following the application makespan decrease. Moreover, the exact amount of bandwidth allocated is taken into account, thus showing a decrease on the data transfer cost for higher performing strategies. This cost is closer to a real measurement of the amount of resources consumed on the platform.

Table 4: Comparison with Amazon EC2

Strategy	Makespan	#VM	Execution cost	
			HIPerNet ($\times 10^5$)	Amazon EC2
Naive	67.08min ± 0.10	35	$1.40 \times c_r + 3.68 \times c_b$	$2\text{h} \times 35\text{VMs} \times c'_r + 1\text{GB} \times c'_b$
FIFO	46.88min ± 0.78	35	$0.98 \times c_r + 1.10 \times c_b$	$1\text{h} \times 35\text{VMs} \times c'_r + 1\text{GB} \times c'_b$
Optimized (without grouping)	37.05min ± 0.25	31	$0.69 \times c_r + 0.98 \times c_b$	$1\text{h} \times 31\text{VMs} \times c'_r + 1\text{GB} \times c'_b$
Optimized (with grouping)	22.93min ± 0.58	31	$0.42 \times c_r + 0.48 \times c_b$	$1\text{h} \times 31\text{VMs} \times c'_r + 1\text{GB} \times c'_b$

4.3.5 Impact of bandwidth control on application cost

Further experiments to evaluate the bandwidth control mechanism were also performed. The application was executed using the *optimized* strategy with service grouping under two additional network bandwidth configurations: lower and higher bandwidth values than the optimal found were tested (1 Mbps and 10 Mbps respectively). Table 5 displays for each configuration: the data transfer time in each stage (in seconds), the application makespan (in minutes) and the corresponding cost. Comparing the results with the optimized bandwidth allocation, it appears that using a low bandwidth, the makespan increases as expected. However, the cost increases as well because the cost gain on network bandwidth is compensated by the loss on computing nodes reservation time. With the high bandwidth, the application makespan can be reduced (-22.72% in this case) at a higher cost (+102% computed with $c_r = c_b = 0.10$).

Table 5: Bandwidth control mechanism evaluation

Bandwidth	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Makespan (min)	Execution cost ($\times 10^5$)
Low (1 Mbps)	222.59 ± 2.51	316.57 ± 40.37	2.91 ± 0.50	34.78 ± 0.67	$0.65 \times c_r + 0.31 \times c_b$
Optimized	53.8 ± 4.56	171.72 ± 24.66	1.53 ± 0.23	22.93 ± 0.58	$0.42 \times c_r + 0.48 \times c_b$
High (10 Mbps)	30.79 ± 3.85	42.68 ± 9.55	1.09 ± 0.18	17.72 ± 0.23	$0.33 \times c_r + 1.61 \times c_b$

5 Related work

This work is related to workflow scheduling, resources management and mapping workflows onto resources. Many existing resource allocation and task scheduling strategies for grid applications (*e.g.* [8]) focus on matchmaking algorithms which goal is not to determine an optimal allocation but is limited to identify suitable resources. Workflow-based allocation algorithms [6,7,18,26] can deliver better performances than matchmaking algorithms. However, the objective of these algorithms is to minimize the application makespan and they do not take into account the execution cost on a pay-per-use platform.

The best-effort algorithms such as *Min-Min*, *Max-Min* [25] or *HEFT* [32] focus only on minimizing the application makespan while other QoS constraints algorithms such as *Deadline/Time Distribution* [38] and *LOSS/GAIN* [29] consider a multi-objective scheduling problem. However, all of them do not take into account the link bandwidth for data exchange between workflow services.

In [28], Ramakrishnan *et al* presented a fault tolerance workflow scheduling algorithm to orchestrate multiple workflows on Grid and Cloud infrastructures by duplicating the execution of some workflows to increase the probability of success of individual tasks. This kind of approach, although potentially efficient in reducing execution time, does not consider the infrastructure cost. Other workflow scheduling algorithms under resource allocation constraints have been also proposed [30,36]. In [30], Senkul *et al* presented an architecture for workflow scheduling that considers resource allocation cost and control constraints (*e.g.* co-allocation of tasks on a same resource). It does not take into account resource limitations and heterogeneity. Furthermore, our approach differs as it considers the trade-off between allocation cost and performance.

Silva *et al* presented in [31] a heuristic for resources allocation on utility computing infrastructure. This heuristic optimizes the number of machines allocated to process tasks and speed up the execution within a limitation of budget. However, this heuristic is only suitable for bag-of-tasks problems in which there is no dependence and the communication between tasks.

Within the Service Level Agreements (SLA) context, Dang *et al* presented in [10,11] the resource allocation algorithms to map grid-based workflows onto grid resources. These algorithms try to assign the workflow tasks to grid resources so as to meet the user's deadline and minimize the cost. These algorithms do not take into account the network bandwidth.

Concerning the virtual resources and network description language, new challenges coming from virtualization techniques have to be considered to complement the specification proposed by classical infrastructures [21,33,13]. Some works have proposed specific languages to describe, model, and exchange information on network topologies [22,2,5]. But in addition, it is needed to combine the spatial and temporal aspects of virtual infrastructures. For example, the Open Virtualization Format (OVF) [9] proposes a mechanism to package and distribute software to be run in one or more virtual machines. Already, the Open Cloud Computing Interface Working Group (OCCI-WG) [1] is investigating a solution to interface with Cloud Infrastructures exposed as services. The cloud infrastructures resources (compute, network and storage) are described using a simple key-value-based descriptor format.

These languages are very efficient for their proposal, but none of them meet all the specification requirements in terms of flexibility, expressiveness, reliability, and simplicity, required to achieve an optimal VPXI specification and allocation [23].

The use of virtual grids to simplify application scheduling has been explored in [20]. They propose a descriptive language, vgDL, which enables users to specify an initial description of the desirable resources, resulting in a pre-selected virtual grid corresponding to a simple vgDL description. vgDL proposes three aggregation types to specify the interconnection network: LooseBag, TightBag and Cluster. The approach proposed in VXDL is more comprehensive and allows the definition of the infrastructure's shape through the description and configuration of virtual links.

The approach of controlled virtual network infrastructures, running in parallel over a shared physical network is an emerging idea offering a variety of new features for the network. Cabo [14] proposes to exploit virtual networks for Internet Service Providers, distinguishing them from the physical infrastructure providers, and giving them end-to-end control. HIPerNet shares the same vision but focuses more on distributed computing application and proposes a language to express the infrastructure requirements in capacity, time, and space.

In [4], the authors propose VINI, a virtual network infrastructure that allows several virtual networks to share a single physical infrastructure, in a similar way to HIPerNet. VINI makes the network transparent to the user, representing each component of the network. This being one of our main interests, HIPerNet provides a language, VXDL, to specify the topology of those components. The GENI project [27] aims to build a shared infrastructure for hosting multiple types of network experiments. VXDL can help in the description of slices and HIPerNet is an orchestration framework that suits GENI's requirements.

Similarly to the network elasticity of HIPerNet, DaVinci [19] is a concept proposing also virtual networks adapting to performance objectives. In DaVinci, the network is monitored and the different virtual networks adapt dynamically to the conditions in order to optimize their performance. While this approach is an interesting way to improve virtual network quality, our approach considers virtual network user's as well as substrate provider's interests, coming with a cost model. A user reserves virtual network capacity to a certain cost and has in return performance guarantees for the reserved period of time.

6 Conclusion

This paper proposed strategies to determine a cost/performance trade-off when executing workflow-based distributed applications on a cloud infrastructure. The advanced network bandwidth control capabilities of the HIPerNet middleware are exploited to extend the traditional cloud paradigm to network provisioning. An experimental validation was carried out using a real workflow-based medical application. Results assess the performance of the *optimized* strategy with job grouping optimization. They show the critical impact of network performance on the application. The solution implemented can be exploited by end-users to minimize their costs or service provider to design resources sharing strategies.

Acknowledgements This work is funded by the French National Agency for Research (ANR), program "Calcul Intensif et Simulation", HIPCAL project (http://hipcal.lri.fr), under contract number ANR-06-CIS-005. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (https://www.grid5000.fr).

References

1. Open Cloud Computing Interface Working Group (OCCI-WG). http://www.occi-wg.org/doku.php, 2009.

2. Ron Addie, Stephen Braithwaite, and Abdulla Zareer. Netml: a language and website for collaborative work on networks and their algorithms. In *ATNAC 06*, 2006.
3. Fabienne Anhalt, Guilherme Koslovski, and Pascale Vicat-Blanc Primet. Specifying and provisioning Virtual Infrastructures with HIPerNet. *ACM International Journal of Network Management (IJNM) - Special issue on Network Virtualization and its Management*, 20(3):129–148, May/June 2010.
4. Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. *ACM SIGCOMM Computer Communication Review (CCR)*, 36(4):3–14, 2006.
5. Kyrre Begnum and John Sechrest. The MLN Manual - version 1.0. `http://mln.sourceforge.net/doc/mln-manual.pdf`, November 2009.
6. Luiz Fernando Bittencourt and Edmundo R. M. Madeira. Towards the Scheduling of Multiple Workflows on Computational Grids. *Journal of Grid Computing (JOGC)*, 8(3):419441, September 2010.
7. Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 759–767, 2005.
8. Tracy D. Braun, Howard Jay Siegel, Noah Beck, Lasislau L. Bölöni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. of Parallel and Distributed Computing (JPDC)*, 61(6):810–837, 2001.
9. Simon Crosby, Ron Doyle, Mike Gering, Michael Gionfriddo, Steffen Grarup, Steve Hand, Mark Hapner, Daniel Hiltgen, Michael Johanssen, Lawrence J. Lamers, John Leung, Fumio Machida, Andreas Maier, Ewan Mellor, John Parchem, Shishir Pardikar, Stephen J. Schmidt, Rene W. Schmidt, Andrew Warfield, Mark D. Weitzel, and John Wilson. Open Virtualization Format Specification (OVF). Technical Report DSP0243, Distributed Management Task Force, Inc., February 2009.
10. Minh Quan Dang and Jorn Altmann. Resource allocation algorithm for light communication grid-based workflows within an SLA context. *International Journal of Parallel, Emergent and Distributed Systems*, 24(1):31–48, 2009.
11. Minh Quan Dang and D. Frank Hsu. Mapping Heavy Communication Grid-Based Workflows Onto Grid Resources Within an SLA Context Using Metaheuristics. *International Journal of High Performance Computing Applications (IJHPCA)*, 22(3):330–346, 2008.
12. Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing (JOGC)*, 1(1):9–23, 2003.
13. Freek Dijkstra and Martin Swany. Network Mark-up Language Working Group (NML-WG). `https://forge.gridforum.org/projects/nml-wg`, 2007.
14. Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, 2007.
15. Tristan Glatard, Johan Montagnat, David Emsellem, and Diane Lingrand. A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution. *Future Generation Computer Systems (FGCS)*, 24(7):720–730, July 2008.
16. Tristan Glatard, Johan Montagnat, Diane Lingrand, and Xavier Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *Int. Journal of High Performance Computing and Applications (IJHPCA)*, 22(3):347–360, August 2008.
17. Tristan Glatard, Xavier Pennec, and Johan Montagnat. Performance evaluation of grid-enabled registration algorithms using bronze-standards. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'06)*, October 2006.
18. Wei Guo, Weiqiang Sun, Weisheng Hu, and Yaohui Jin. Resource Allocation Strategies for Data-Intensive Workflow-Based Applications in Optical Grids. In *10th IEEE Singapore International Conference on Communication Systems (IEEE ICCS 2006)*, pages 1–5, October 2006.
19. Jiayue He, Rui Zhang-Shen, Ying Li, Cheng-Yen Lee, Jennifer Rexford, and Mung Chiang. Davinci: dynamically adaptive virtual networks for a customized internet. In *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–12, New York, NY, USA, 2008. ACM.
20. R. Huang, H. Casanova, and A.A. Chien. Using virtual grids to simplify application scheduling. In *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006)*, April 2006.
21. Distributed Management Task Force Inc. Common Information Model (CIM) Standards. `http://www.dmtf.org/standards/cim/`.
22. Xuxian Jiang and Dongyan Xu. vbet: a vm-based emulation testbed. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 95–104, New York, NY, USA, 2003. ACM.
23. Guilherme Koslovski, Tram Truong Huu, Johan Montagnat, and Pascale Vicat-Blanc Primet. Executing distributed applications on virtualized infrastructures specified with the VXDL language and managed

by the HIPerNET framework. In *First International Conference on Cloud Computing (CLOUDCOMP 2009)*, Munich, Germany, October 2009.

24. Guilherme Koslovski, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charão. VXDL: Virtual Resources and Interconnection Networks Description Language. In *GridNets 2008*, Oct. 2008.
25. Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Eighth Heterogeneous Computing Workshop (HCW'99)*, pages 30–44, San Juan , Puerto Rico, April 1999. IEEE Computer Society.
26. Anirban Mandal, Ken Kennedy, Charles Koelbel, Gabriel Marin, John Mellor-Crummey, Bo Liu, and Lennart Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*, pages 125–134, Washington, DC, USA, 2005. IEEE Computer Society.
27. Larry Peterson, Tom Anderson, Dan Blumenthal, Dean Casey, David Clark, Deborah Estrin, Joe Evans, Dipankar Raychaudhuri, Mike Reiter, Jennifer Rexford, Scott Shenker, and John Wroclawski. GENI Design Principles. *Computer*, 39(9):102–105, 2006.
28. Lavanya Ramakrishnan, Daniel Nurmi, Anirban Mandal, Charles Koelbel, Dennis Gannon, T.M Huang, Yang-Seok Kee, Graziano Obertelli, Kiran Thyagaraja, Rich Wolski, Asim YarKhan, and Dmitri Zagorodnov. VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC09)*, November 2009.
29. Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D. Dikaiakos. Scheduling Workflows with Budget Constraints. In *CoreGRID Integration Workshop (CGIW2005)*, pages 347–357, Pisa, Italy, November 2005. Springer-Verlag.
30. Pinar Senkul and Ismail H. Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, 30(5):399–422, 2005.
31. João Nuno Silva, Luís Veiga, and Paulo Ferreira. Heuristic for resources allocation on utility computing infrastructures. In *6th International Workshop on Middleware for Grid Computing (MGC 2008)*, pages 1–6. ACM, December 2008.
32. H. Topcuoglu, S. Hariri, and Wu Min-You. Performance-effective and low-complexity task scheduling for heterogeneous computing. *International Journal of Supercomputer Applications (IJSA)*, 13(3):260–274, March 2002.
33. Jeroen van der Ham, Paola Grosso, Ronald van der Pol, Andree Toonk, and Cees de Laat. Using the network description language in optical networks. In *Proc. IFIP/IEEE IM*, May 2007.
34. Pascale Vicat-Blanc Primet, Fabienne Anhalt, and Guilherme Koslovski. Exploring the virtual infrastructure service concept in Grid'5000. In *20th ITC Specialist Seminar on Network Virtualization*, Hoi An, Vietnam, May 2009.
35. Pascale Vicat-Blanc Primet, Vincent Roca, Johan Montagnat, Jean-Patrick Gelas, Olivier Mornard, Lionel Giraud, Guilherme Koslovski, and Tram Truong Huu. A Scalable Security Model for Enabling Dynamic Virtual Private Execution Infrastructures on the Internet. In *IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2009)*, Shanghai, China, May 2009.
36. Zhijiao Xiao, Huiyou Chang, and Yang Yi. *Optimization of Workflow Resources Allocation with Cost Constraint*, pages 647–656. Springer Berlin / Heidelberg, August 2007.
37. Jia Yu and Rajkumar Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing (JOGC)*, 3(3-4):171 – 200, September 2005.
38. Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-Based Scheduling of Scientific Workflow Application on Utility Grids. In *First International Conference on e-Science and Grid Computing (E-SCIENCE'05)*, pages 140–147, Melbourne, Australia, December 2005. IEEE Computer Society.
39. Henan Zhao and Rizos Sakellariou. Scheduling Multiple DAGs onto Heterogeneous Systems. In *15th Heterogeneous Computing Workshop (HCW 2006)*, Rhodes Island, Greece, April 2006.