Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure

Tram Truong Huu, Johan Montagnat

Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure

Tram Truong Huu
University of Nice - Sophia Antipolis
I3S laboratory, France
Email: tram@polytech.unice.fr

Johan Montagnat
CNRS, I3S laboratory
Sophia Antipolis, France
Email: johan@i3s.unice.fr

***Abstract*—Cloud computing infrastructures are providing resources on demand for tackling the needs of large-scale distributed applications. Determining the amount of resources to allocate for a given computation is a difficult problem though. This paper introduces and compares four automated resource allocation strategies relying on the expertise that can be captured in workflow-based applications. The evaluation of these strategies was carried out on the Aladdin/Grid'5000 testbed using a real application from the area of medical image analysis. Experimental results show that optimized allocation can help finding a trade-off between amount of resources consumed and applications makespan.**

***Index Terms*—cloud infrastructures, resources allocation, workflows**

I. 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 heavy-weight applications by adapting the execution environment to the specific application requirements. Furthermore, business models have been developed to determine the infrastructure cost for a specific, metered usage.

From a user perspective, 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. Resources could be allocated on demand, during the application execution, but resources deployment is a time-consuming process that impacts the application performance. 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 qualitative estimation. Workflow-based applications represent a large class of coarse-grained distributed applications [6]. Taking advantage of the workflow formalism, the application logic can be interpreted and exploited to produce an execution schedule estimate.

The objective of this paper is to design virtual infrastructures allocation strategies for cloud computing platforms which size and topology are optimized according to some user-controlled metric, using the application expertise captured by the workflow representation. Four strategies are proposed and evaluated through experiments involving a real application in the area of medical image analysis. The Aladdin/Grid5000 research infrastructure provides a substrate for the virtual infrastructures allocation.

II. Cost model for workflow-based applications

A workflow application is defined through a workflow graph featuring the application services to be executed (workflow nodes) and the dependencies between these services (edges). An example application workflow is shown in Figure 5. 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. The role of a workflow engine is to scale the execution for a specific input data set. 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.

The trade-off between an execution infrastructure cost and the application performance is optimized using a cost function which parameters depend on the allocated infrastructure size. Both computing resources and network bandwidth are considered in the cost function defined below. As will be discussed later, only acyclic workflows for which the execution schedule can be statically determined are considered. In our approach an execution can occur in several *stages*. For each stage, a given-size infrastructure is allocated to perform the execution of part of the workflow during a period. Each infrastructure redeployment, between different stages, is time-consuming.

One extreme condition, is to make a single reservation 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.

Commercial cloud infrastructures use a simple cost computation model (*e.g.* Amazon EC2[1] charges users per day of resources usage) and let the user responsible for precisely estimating the amount of needed resources. The model proposed below makes a finer grain estimate of the real infrastructure usage which helps the user estimating the exact amount of resources that will be consumed for each run of an application. After finishing the execution, allocated resources are returned to the cloud infrastructure. Let m_{max} be the maximum number of resources 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 resources used in each execution stage with $\forall i, m_i \leq m_{max}$. If we assume the per-unit cost of a resource is c_r, then the total computing cost of the infrastructure allocated for the application is:

$$C_r = c_r \sum_{i=1}^{s} m_i \left(T_{di} + T_i(m_i, n, b)\right) \tag{1}$$

where T_{di} and $T_i(m_i, n, b)$ is the deployment time and execution time of stage i, respectively. T_i depends on the number of resources reserved for this stage (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 exchanges. The total infrastructure cost is also impacted by the data transfer time. If the per-unit cost of the reserved bandwidth is c_b, then the total data transfer cost is:

$$C_b = \begin{cases} c_b \sum_{i=1}^{s} \sum_{j=1}^{k_i} t_j b_j & \text{(2a)} \\ c_b \sum_{i=1}^{s} \left(T_{di} + T_i(m_i, n, b)\right) \sum_{j=1}^{k_i} b_j & \text{(2b)} \end{cases}$$

where t_j is the effective data transfer time on link j. Case 2a applies if the infrastructures charges network usage according to the amount of data transferred (*e.g.* Amazon EC2). Case 2b applies if the infrastructure can allocate controlled bandwidth and charge network usage according to the total time of reservation (*e.g.* HIPerNet [9]).

From formulas 1 and 2, the total infrastructure cost to execute the application is $C = C_r + C_b$. This cost has to be optimized considering a maximum admissible cost and the application performance scalability. C depends on the value of T_i at each execution stage. The computation of T_i is possible using the application logic described through the workflow. The workflow engine used, MOTEUR [6], usually produces an execution schedule and controls 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 links bandwidth is available.

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

III. Virtual resources allocation strategies

The application execution time for each stage (T_i) depends on the resources allocated for execution by the scheduler. Four strategies are described below to allocate resources and schedule computing tasks on these.

A. 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 virtual computing resources may be allocated and naively split: $mt_i / \sum_j t_j$ resources are dedicated to each service i. The network bandwidth is similarly allocated proportionally to the amount of data to transfer between each pair of services. 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 invoked during the whole duration of the workflow execution. It serves as a performance base-line.

B. 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, figure 1 displays the estimated execution time and the total cost of the workflow from figure 5 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. This value is much smaller than bandwidth capacity of the cloud infrastructure which however must have a mechanism to share the link for other users without interfering with each other.

C. 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 execution graph (DAG), using the second composition approach presented in [14] for instance. Secondly, the DAG is divided in

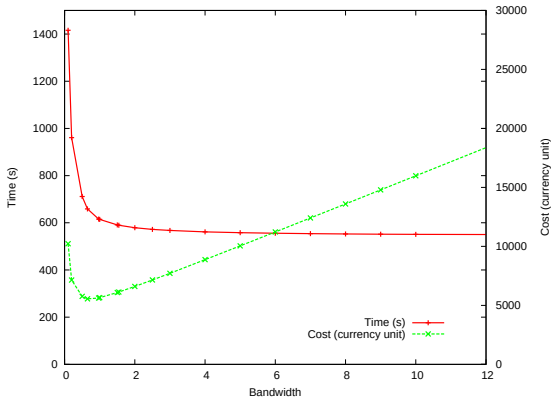


Fig. 1: Estimation of the execution time and total cost with regard to the bandwidth of the *FIFO* strategy

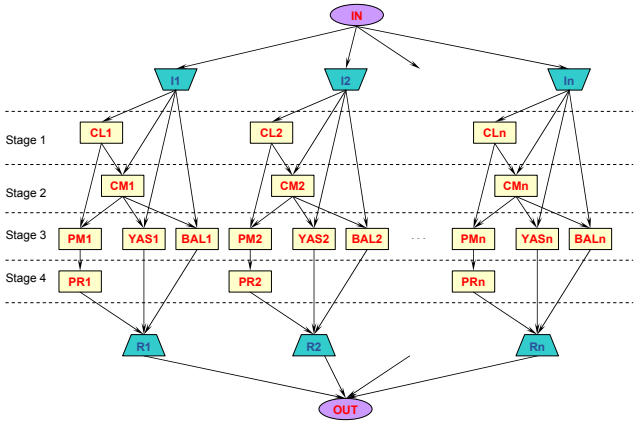


Fig. 2: DAG jobs of Bronze Standard application for n inputs

execution stages, each of them meant to be executed on a specific virtual infrastructure. An example execution DAG for the workflow of figure 5 is shown in figure 2, 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). This represents a broad category of workflows in e-Science (many data-intensive, scientific workflow languages do not support loops).

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 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 figure 2), m_{max} data items are processed concurrently by S and the process is repeated n/m_{max} times, leading to the execution time:

$$T_i = \begin{cases} [\frac{n}{m_{max}}] * T_S & \text{if } n \bmod m_{max} = 0; \\ \left([\frac{n}{m_{max}}] + 1\right) * T_S & \text{otherwise} \end{cases} \quad (3)$$

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:

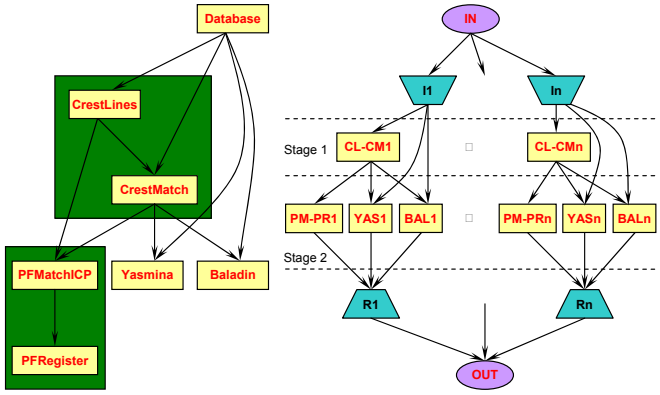
$$T_j = \begin{cases} [\frac{inv_j}{m_j}] * T_{uj} & \text{if } inv_j \bmod m_j = 0; \\ \left([\frac{inv_j}{m_j}] + 1\right) * T_{uj} & \text{otherwise} \end{cases} \quad (4)$$

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

D. 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 [5]. 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 figure 5, two services groups are identified which do not cause loss of parallelism as shown in figure 3a. The number of execution stages can also be reduced as shown in figure 3b.

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



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

Fig. 3: Services grouping without parallelism loss

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 figure 4.

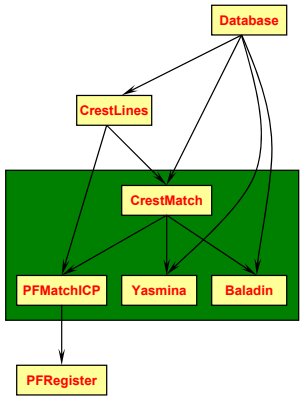


Fig. 4: Grouping CrestMatch, PFMatchICP, Yasmina and Baladin

IV. Validation on the Aladdin/Grid'5000 testbed

A. Test application

The experiments are performed using the *Bronze Standard* (BS) a real workflow-based application from the area of medical image analysis [7]. 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 figure 5. In the experiments reported below, a clinical database with 59 pairs of patient images was used. For each run, 354 computing tasks were generated.

B. 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, and the MOTEUR workflow engine was created. The infrastructures allocated are managed by the HIPerNet virtual infrastructure deployment middleware[2] [9]. HIPerNet enables the joint virtualization of computing and network resources. Consequently, our experiments use equation 2b to compute C_b. The physical resources were reserved on the fully reconfigurable Aladdin/Grid'5000 research infrastructure[3], clusters *helios* and *sol* in Sophia Antipolis, France. It is to be noted that the Aladdin/Grid'5000 infrastructure and the HIPerNet virtualization layer currently do not enable the control of bandwidth between pairs of nodes although this is a planed extension. The physical resources are Sun Fire X2200 M2 machines, 2.6GHz, 4 cores and 4GB RAM interconnected through 10Gbps Ethernet. One virtual machine is deployed per physical machine. For all experiments, 35 physical machines were reserved, 3 of which are dedicated to the central services. The 32 machines left were allocated to application services.

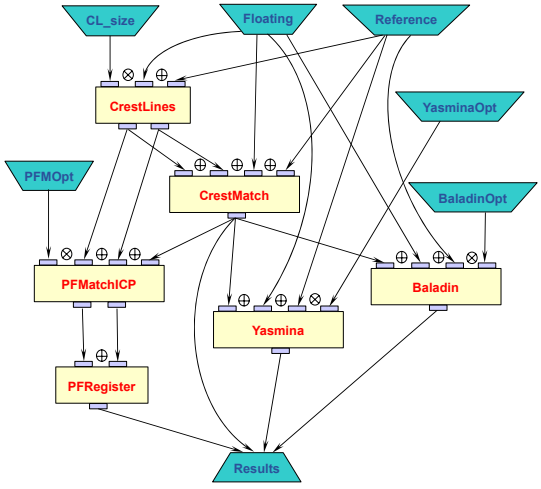


Fig. 5: Bronze Standard workflow.

Services	Time (seconds)	Input data	Produced data
CrestLines	31.06 ± 0.57	15MB	10MB
CrestMatch	3.22 ± 0.51	25MB	0.2MB
PFMatchICP	10.14 ± 2.41	10.2MB	240kB
PFRegister	0.64 ± 0.22	240kB	160kB
Yasmina	52.94 ± 2.96	15.2MB	0.2MB
Baladin	226.18 ± 19.36	15.2MB	0.2MB

TABLE I: Benchmark of the BS services execution time and data transfer volumes.

For the needs of the MOTEUR planner, all 6 services involved in the BS workflow have been benchmarked for

[2] http://www.ens-lyon.fr/LIP/RESO/Software/hipernet/index.html
[3] http://www.grid5000.fr

execution time and amount of data transferred as reported in table I. Data transfers are performed through the secure copy protocol (*scp*). The scp connection establishment time between two virtual machines proved to be non-negligible (seconds) and it was taken into account by adding it to the service effective computation time. For each experiment, the application was executed 5 times and the makespan was averaged to minimize the side-effect of other grid users activity on these measurements. The standard deviation is also reported.

For each strategy, the planner optimizer was executed to determine the configuration with the minimal execution time. The *naive* and *FIFO* strategies are single-stage and they consistently minimize execution time when all 32 resources are allocated. 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.

The *naive* allocation strategy allocated the 32 computing nodes to each services as follows: 3 nodes for CrestLines, 1 node for CrestMatch, 1 node for PFMatchICP, 1 node for PFRegister, 4 nodes for Yasmina, and 22 nodes for Baladin. The application makespan is 60.35min $\pm$ 0.1min. This experiment shows that the virtual resources are not well exploited during the execution. Figure 6 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, darker 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.

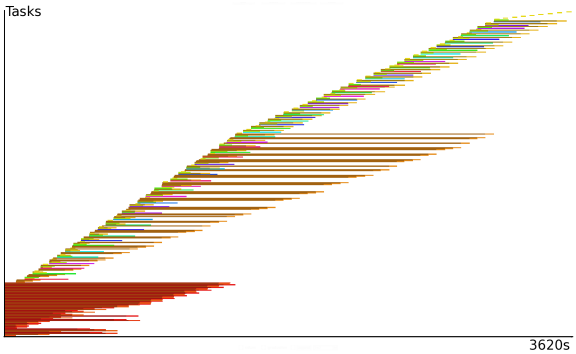


Fig. 6: Tasks schedule with the *naive* strategy

The makespan of the *FIFO* strategy is much improved: 21.10min $\pm$ 0.5min. 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. Firgure 7 shows a typical task schedule for this strategy.

For the *optimized* strategies, the planer determines the number of virtual resources and the bandwidths yielding to a minimal execution time. Without services grouping there are 4 execution stages. According to the optimization results: only 30 nodes were allocated for the first, second and fourth stages (additional resources would be wasted). The bandwidth is 3.25Mbps, 2.95Mbps and 0.54Mbps, 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 0.69Mbps, 0.76Mbps and 0.80Mbps, respectively. Although we cannot yet control the bandwidths on the experimental testbed, the values found are supported by the physical links. The application makespan is then 21.70min $\pm$ 0.25min. Further grouping the application services as shown in figure 4, the application is divided into three stages only, using 30 nodes each. The bandwidth allocated for each stage is 3.25Mbps, 0.94Mbps and 0.54Mbps, respectively. The application makespan is then 18min $\pm$ 0.3min. Besides the execution time improvement, the number of resources consumed is also lowered. As we can observe in figure 8, 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.

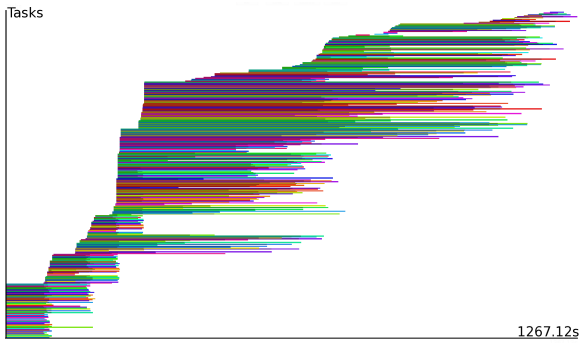


Fig. 7: Tasks schedule with the *FIFO* strategy

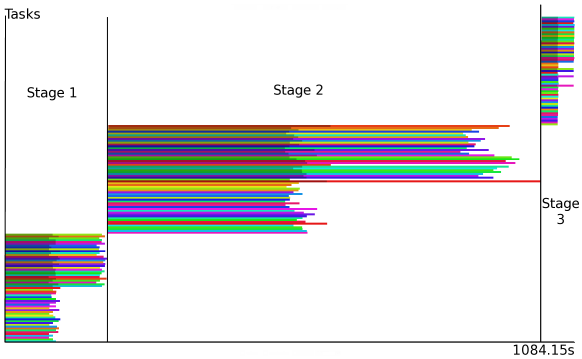


Fig. 8: Tasks schedule with *optimized* services grouping

In conclusion, table II compares the performance of the strategies presented above. The worst case is the *naive* strategy that uses the maximum number of resources for a very large makespan. The *FIFO* and *optimized* strategy without grouping services have approximately the same application makespan but the *optimized* strategy uses less resources than *FIFO*. The

best case is the *optimized* strategy with grouping services, it uses less resources and returns the results the most rapidly.

Strategy	Makespan	No. Resources
Naive	60.35min $\pm$ 0.1min	32
FIFO	21.10min $\pm$ 0.5min	32
Optimized (without grouping)	21.70min $\pm$ 0.25min	30
Optimized (with grouping)	18min $\pm$ 0.3min	30

TABLE II: Performance comparison between four strategies

V. 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.* [2]) focus on matchmaking algorithms that do not search for an efficient allocation. Workflow-based allocation algorithms [1], [8], [10] 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.

In [11], 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 [12], [13]. In [12], 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.

Within the Service Level Agreements (SLA) context, Dang *et al* presented in [3], [4] 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.

VI. Conclusion and future works

This paper proposed a cost-based approach for allocating resources to workflow-based applications. It defines virtual infrastructure allocation strategies and presents associated experiments using a real workflow-based medical application. Experimental results assess the performance of the *optimized* strategy and job grouping optimization. Based on these promising results, our future works will explore an approach to automate the translation of the workflow into a virtual resources description language in order to externalize the management of the infrastructure to the cloud middleware.

Acknowledgement

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] 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.

[2] 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. *Journal of Parallel and Distributed Computing (JPDC)*, 61(6):810–837, 2001.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Pinar Senkul and Ismail H. Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, 30(5):399–422, 2005.

[13] Zhijiao Xiao, Huiyou Chang, and Yang Yi. *Optimization of Workflow Resources Allocation with Cost Constraint*, pages 647–656. Springer Berlin / Heidelberg, August 2007.

[14] Henan Zhao and Rizos Sakellariou. Scheduling Multiple DAGs onto Heterogeneous Systems. In *15th Heterogeneous Computing Workshop (HCW 2006)*, Rhodes Island, Greece, April 2006.