Cloud Resource Management Using Constraints Acquisition and Planning

Yannick Le Nir and **Florent Devin** and **Peio Loubière**
26 avenue des Lilas 64062 Pau Cedex 9

Abstract

In this paper we present a full architecture to deploy efficiently a grid in a private cloud approach. We first give details about the resources constraints acquisition. We use Rich Internet Application (RIA) to access and/or modify the resources in a very user-friendly interface. Then, using the previous information, we explain how we can compute a dynamic deployment plan, that can be used either to build an optimal grid of computers or to give information to its scheduler. This plan is computed using pddl solver with various logical constraints obtained from the IT users through the RIA.

Introduction

Nowadays, the historic term grid computing is being changed into the marketing term cloud computing. Behind this terminological variation, there is a real evolution that impacts a lot of uses. Whereas grid computing often has specific middle ware, in cloud computing it is now very useful to work with scalability and heterogeneous software and hardware. Thanks to the simple way to deploy application in the cloud, we should completely reconsider the resources management. Researchers or other users no longer have to consider initial challenges such as the way to run a job, or to transfer files, nor the management of many users on many systems. They can concentrate on the most important task being the description of their problem in an efficient language which must be very simple, powerful and without any learning curve. The integration of this process in an existing IT is essential to reduce the data redundancy and to maintain consistent information. While cloud computing is becoming almost commonplace, an essential challenge is then the resources management both from users and data-centers. In this paper, we first present a simple architecture built as a SOA to resolve the resource management problem. This architecture is divided into two services. The first one is a specific controller dedicated to solving the management task. The second service is a rich web interface plugged into the existing IT. Both offer a powerful solution to the dynamic resource management problem for many cloud computing architecture deployments.

Cloud resource management

Grid computing

During the 1990s, many projects started using the term grid for distributed computing architecture. Initially a major goal was to enable resources sharing for scientific collaborations. In (**?**), they argue that the use of such architectures is relevant when a common set of requirements for *resource sharing and problem solving in dynamic, multi-institutional collaborations* exists. Due to their distributed and often ephemeral nature, the term *virtual organization* is sometimes used. Behind this term, we found various applications that today form the basis of most famous cloud based applications.

From grid to cloud

In this case, what are the main differences between the historic term grid computing and the fashionable one cloud computing (**?**),(**?**)? Many people argue that there is no fundamental difference, and consider it is just a terminological evolution. In (**?**), it is said that *This is not a technology game but a change-management game*. However, we can find some other important evolutions, that we will consider in our work:

- the present cloud computing is deployed in nets where transmissions are more efficient between computers (**?**).
- the progress of virtualizations, due to the increased power of computers, now offers a real solution for every application deployment in distributed architectures whatever the heterogeneity of software (**?**).
- the necessity to reconsider controls and resource management in a more flexible way. Indeed, as a consequence of the two previous points, we can now consider a cloud computing that can be easily accessed by anyone who has no technical background (**?**).

Before giving details about cloud management specificities, we have to finish the context description of our work including an important difference between the so called public and private cloud (**?**). It is a highly debated question to decide if a cloud based on private infra-structures has a sense. Most of the time, a private cloud is deployed in a unique existing IT that is in charge of rights and resources management. Private cloud can be seen as simulation of cloud computing on

private networks. The ability to scale up or down depending on demand for such networks is a similar challenge as in classic public cloud (**?**). In a company, there is often a set of computers available. In this case, the objections related to the equipement cost are not relevant since companies need their own computers.

Later in this paper, we will concentrate on such private cloud. Indeed, we will use the fact that we know the complete architecture of the IT to generate deployment plans that match the real management of resources optimally.

Cloud management specificities

Within the previous description of a private cloud, we can expose the management specificities that we will consider in our deployment plan. To deploy computers on the grid which will build the private cloud we have a multicriteria approach. This is the first specificity.

The different criteria we take into account mix the conflicting goals to maximize the amount of available resources and to minimize the energy consumption. Such a problematic is common in classic cloud (**?**) and can be improved in our model. The second specificity is the dynamic behavior of resources since the elements of the grid are usual and non specific computers. The third and last specificity is the separation of constraints in two distinct classes, called hard and soft constraints. These constraints give indications to the planning without blocking the deployment phase.

Problem modeling

The main goal of our application is to build a deployment plan for all existing resources that can be integrated into the grid. Thus we have to collect information describing the availabilities of these resources and the needed amount of computing time queried by users during a fixed period. These two parts are essential and often not efficiently integrated in classic architecture. The choice we made to consider only private cloud is here a real advantage. Indeed, we can reuse the existing IT to collect already known information about the state of the resources that could be integrated into the grid. This resource constraints acquisition is done using a Rich Internet Application that communicates with the IT and gives facilities for users to interact with the private cloud. Once we have collected all the needed information, we can translate it into logical constraints that will be used by a solver to build a deployment plan related to time-line. This plan can be used to build a dynamic and efficient grid that really meets the requirement of the users. This computational part is done using the famous description language pddl and classic efficient solvers such as FF.

Resource constraints acquisition

In public cloud, you do not have to deal with availability. It offers a service and you "just" have to pay to use this service. Regardless of whether there are enough computers to satisfy your demand on time t, you simply want to have a huge computing capacity. But in a private cloud, you do not have to pay (you have already done it, by buying computers). You can not simply ask the system to have enough processors to meet your needs. You have to consider many aspects. Some of these are functional (what kind of computer you have), others are more preferences (like green computing), and others are structural (computer might be under maintenance).

If the first two categories can be handled by the system deploying the grid, as we have seen previously, the latter have to be done before running the cloud manager. In fact, structural availabilities have to be considered as an entry for the cloud management process. For now, we have considered three different kinds of structural availabilities:

- computer unavailability: means that the usual user is working on it;
- computer maintenance: means that the system administrator has to do some work on the computer;
- computer unreachable: means that the room in which the computer is located is under maintenance.

Once you have a tool to facilitate the "constraints acquisition", you can easily transform them into time slots constraints entry.

Application

Although our model can be used in every company that wants to retrieve unused computing time, it is very well adapted for a university or engineering school that owns a lot of computers with very flexible use. We will take for example our engineering school for which we have designed a new software to automatically generate timetables : Stratus (**?**; **?**). This tool deals with several constraints. Dealing with all these constraints implies that the timetable could often change. With this software, we are able to extract unavailability of each computer science room. But this is not sufficient to have an efficient tool to deploy a private grid. As we schedule classes into classrooms, we are also able to know how many computers are used. So for each time of each day we are able to find out exactly the number of free computers in each room.

As we are in a school, we have to provide a tool to inform all users that a particular room is unavailable. The administration, in charge of dealing with these events, can put a hard constraint on the necessary room, to prevent Stratus generating a flimsy timetable. Putting a hard constraint on Stratus is very easy, and it can be done in two ways:

- Using a particular Google calendar and scheduling a room reservation, like anybody would schedule an appointment on his own Google calendar.
- Using Stratus and mark a reservation for the room. In Stratus this can be done using a visual interface (like Google calendar), or by using a specific textual interface.

As a computer scienc eengineering school, we have to provide up-to-date computers (for those who do not use laptops). Providing up-to-date computers involves various tests (configurations, software testings, scalable test, ...). So the system administrators can use several computers. So we

have to provide a particular way to handle this. Usually system administrator are people that often prefer textual interfaces rather than graphical ones. With this in mind, we have developed a specific web service to allow the administrator to use a textual interface. He can use this web service to make a reservation, or send a mail (which is closer to a form than a mail) to a specific user. Sending this mail will be automatically translated as a call to the web service. He can also use the same facilities as previously mentioned. But as he can use only part of a room, there is a particular field to indicate how many computers of a room (a number, or a percentage) he needs.

Figure **??** shows how the system handles all kinds of constraints to provide a complete description of the use of all computers.

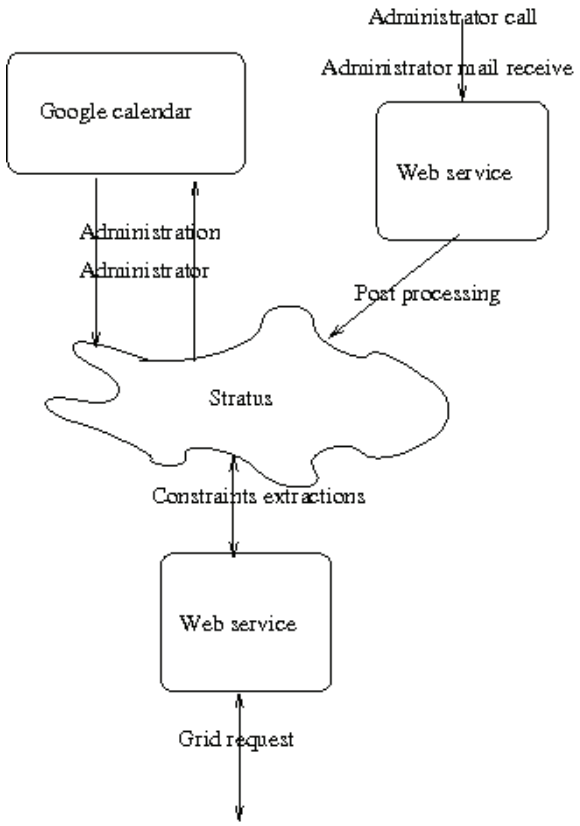

Figure 1: Constraints acquisition - export

Deployment planning

Once we have collected the resources and users'constraints for a fixed time period, we can compute a deployment plan. In this part, we explain the process of computer allocation to the grid during several time slots under constraints.

Constraints

If we consider our particular case, we have several computer rooms that are used by students during several time slots per week. We spend a lot of time with many computers on but with no particular use.
To optimize the use of these computers, we want to construct, in a dynamic way, a grid according to their availability, the electric consumption, etc. in order to execute jobs in a private cloud architecture. It is a fixed slot problem, a week is divided into many equal time slots, the duration of a single time slot is a parameter. We also consider that we rent computing time for jobs to be executed in our cloud; this time is expressed in numbers of time slots.
To manage this problem, we must consider the following items : computers to include in the grid, unavailable time slots, the jobs to be planned in the grid and the number of time slots needed for each job.
Resolving this problem means associating a job's slot with a computer at an available slot. This problem leads us to consider some constraints that the planning process must validate. We separate them into two categories : hard and soft constraints. Hard constraints are constraints we must satisfy without exception, soft constraints are constraints we want to respect as far as possible but which might not be satisfied
The hard constraints are :

- a computer must be added to the grid if a job needs it;
- a computer can be added to the grid at a certain time slot if it is available during this time slot.

The soft constraints are :

- to optimize the cost of running many computers, a maximum number of jobs should be running during night slots (night preferred);
- the job execution must not be spread throughout the week; as far as possible, we must plan jobs in following available slots (following slots);
- we can also define one job's priority for those which are urgent and are therefore the first planned (job priority);
- we can estimate a night and day number of computers available to control electricity consumption (from computers or cooler system, for example) (max computers);
- we can also determine if a job needs a minimum computing capacity, to select only the computers which fit (min capacity).

Planning description using PDDL

This problem is a typical scheduling problem as defined in (**?**) : "exact allocation of activities to resources (...) over time respecting precedence, duration, capacity and incompatibility constraints". In our model, the objects are quite simple and the constraints are limited, thus, we choose PDDL as planning definition language to describe our problem.
First we must define the objects involved in the scheduling : a `job`, on a `computer` at a `time slot`.

Then we defined predicates and a function to satisfy hard constraints:

- `(exec ?j ?slot)` which specifies if a job is already being processed in the grid at a certain time slot (no job parallelization yet);

- `(add-grid ?comp ?slot)` which specifies that a computer is added to the grid at a certain time slot;
- numeric function `(needed-slot ?job)` to evaluate how many slots are dedicated to a job (slots are distinct).

We must also satisfy the defined soft constraints:

- night preferred :
  $j \in job, \exists s \in timeslot,\ available(s) \wedge is-night(s)$
- following slots :
  $s0 \in timeslot,\ avail(s0),$
  $\forall s \in timeslot, avail(s) \wedge s0 \leq s$
- max computers :
  $s0 \in timeslot,\ max-computers(s0) > 0$
- job priority :
  $j0 \in job,\ neededslot(j0) > 0,\ \forall j \in job,\ j \neq j0 \wedge$
  $neededslot(j) > 0 \wedge priority(j0) \leq priority(j)$
- min capacity :
  $j0 \in job,\ \exists c \in computer,$
  $mincapacity(j0) \leq capacity(c)$

To implement them, we used numeric functions (such as (min-capacity ?job), (job-priority ?job) or (slot-weight ?slot), to include a precedence of time slots), predicates (like (is-night ?slot), (used ?comp)) and existential preconditions to manage difference between night and day slots, priority, consecutiveness etc.

Solver

Using PDDL description, we could improve several available solvers from past planning competitions. Considering our PDDL description, the language version being 2.1 (by the use of fluent functions) as defined in (**?**), our choice reduced the numbers of available solvers. First we chose lpg (**?**) and lpg-td (**?**) solvers because we had metrics to optimize and we left soft constraints management to the metric optimization. However we gave up because we had many optimizations and we didn't find the right criteria, and above all, the results were optimized but they depended from the starting seed, and it was not as good as we expected in all cases. So we chose Metric-FF (**?**), which does not provide metric optimization, but whose solution is much more precise, considering the fact that we have expressed all soft constraints in PDDL functions and predicates.

Results

To illustrate the work on the solver, here are excerpts from our PDDL domain and problem files and the computed solution. The domain file (Figure **??**) is composed of : numeric values (lines 5-9), predicates (lines 12-14) which describe hard and soft constraints. Then the main action function to be excecuted : plan (line 16). This function must verify all preconditions (lines 19-26) (still computer available for current slot, current computer not used, with enough capacity for current job and there is no other job with higher priority yet to plan), and the effects of post-conditions (effects, lines 27-30, add the computer to the grid at current slot, check the job is in execution at current slot (no parallelization) and decrease the number of slots to plan for this job, and the

```
1  (define (domain grid-domain)
2  ...
3   (:types timeslot computer job)
4   (:functions
5     (slot-weight ?c - timeslot)
6     (max-computers ?c - timeslot)
7     (comp-capacity ?m - computer)
8     (job-priority ?j - job)
9     (needed-slot ?j - job)
10 ...)
11  (:predicates
12    (add-grid ?m - computer ?c - timeslot)
13    (is-night ?c - timeslot)
14    (exec ?j - job ?c - timeslot)
15 ...)
16  (:action plan
17   :parameters (?j - job ?c - timeslot ?m - computer )
18   :precondition (and
19     (> (max-computers ?c) 0)
20     (not (add-grid ?m ?c)) ...
21     (≥ (comp-capacity ?m) (min-capacity ?j))
22   ...
23    (not (exists (?jb - job)
24       (and (< (job-priority ?j) (job-priority ?jb))
25         (> (needed-slot ?jb) 0))))
26   ...)
27   :effect (and
28     (add-grid ?m ?c) (exec ?j ?c)
29     (decrease (max-computers ?c) 1)
30     (decrease (needed-slot ?j) 1))
31 ))
```

Figure 2: PDDL domain file

number of computers composing the grid at this slot).
The PDDL problem file (Figure **??**) lists all the objects involved, the facts they verify, the initialization of numeric values and the goal to achieve. In our example, our problem is defined by :

- 18 slots available (line 4; in 27 possible, a slot is 6h long, 1^{st} one, (ts0) is Monday 0 to 6h, then it follows), with hierarchy (line 9), night slots defined (line 12) and maximum 3 computers authorized per slot (line 13)
- 70 computers (but 3 used at the same time), with capacity defined (lines 14-15), 4 or 8Ghz, for example)
- 12 jobs to plan, each one has a number of slots (line 10), a priority (n^o 7, 9 and 1 are the 3 most urgent jobs to plan), and a minimum computing capacity (lines 15-16, 8Ghz for all)
- the goal (lines 18-19) is for each job to have all its slots planned

A solution found by Metric-FF (Figure **??**), we can see that the priority is respected, the earlier night slot is given to the first job, then the next slot, etc. The maximum computers per slot is also respected and the capacity too (computer 67 and 66 are not used)

Conclusion

In this work, we have proposed a complete architecture to extract availabilities from a set of computers in an existing IT and to compute a deployment plan for the grid in a private cloud approach. This plan (Figure **??**) associates computers to time slots and will be used to dynamically add or retract computers from the grid. This plan can now be used by many grid tools such as Gridgain or Diet. The integration is easy since it only needs to integrate the acquisition part

```
(define (problem grid-pb)
(:domain grid-domaine)
(:objects
  ts27 ... ts8 ts7 ts4 ts3 ts0 - timeslot
  comp0 ... cmp68 cmp69 - computer
  job0 ... job11 - job
)
(:init
  (= (slot-weight ts0) 0) ...
  (= (needed-slot job1) 3) ... (= (needed-slot job7) 2) (= (needed
      -slot job9) 3)
  (= (job-priority job1) 100) (= (job-priority job7) 200) (= (job-
      priority job9) 150)
  (is-night ts0) (is-night ts4) (is-night ts8)...
  (= (max-computers ts0) 3) (= (max-computers ts3) 3)...
  ... (= (comp-capacity cmp65) 8) (= (comp-capacity cmp66) 4)
  (= (comp-capacity cmp67) 4) (= (comp-capacity cmp68) 8)
  (= (min-capacity job0) 8)...
)
(:goal
  (and (= (needed-slot job0) 0) ...(= (needed-slot job11) 0))
))
```

Figure 3: PDDL problem file

```
 0: PLAN JOB7 TS0 CMP69
 1: PLAN JOB7 TS3 CMP69
 2: PLAN JOB9 TS0 CMP68
 3: PLAN JOB9 TS3 CMP68
 4: PLAN JOB9 TS4 CMP69
 5: PLAN JOB1 TS0 CMP65
 6: PLAN JOB1 TS3 CMP65
 7: PLAN JOB1 TS4 CMP68
 8: PLAN JOB3 TS4 CMP65
 9: PLAN JOB3 TS7 CMP69
10: PLAN JOB5 TS7 CMP68
11: PLAN JOB5 TS8 CMP69
12: PLAN JOB5 TS11 CMP69
13: PLAN JOB5 TS12 CMP69
14: PLAN JOB5 TS15 CMP69
15: PLAN JOB5 TS16 CMP69
16: PLAN JOB5 TS19 CMP69
17: PLAN JOB2 TS7 CMP65
18: PLAN JOB2 TS8 CMP68
19: PLAN JOB2 TS11 CMP68
20: PLAN JOB2 TS12 CMP68
21: PLAN JOB2 TS15 CMP68
22: PLAN JOB2 TS16 CMP68
23: PLAN JOB2 TS19 CMP68
24: PLAN JOB2 TS20 CMP69
25: PLAN JOB8 TS8 CMP65
26: PLAN JOB8 TS11 CMP65
27: PLAN JOB8 TS12 CMP65
28: PLAN JOB11 TS15 CMP65
29: PLAN JOB11 TS16 CMP65
30: PLAN JOB10 TS19 CMP65
31: PLAN JOB6 TS20 CMP68
32: PLAN JOB6 TS21 CMP69
33: PLAN JOB6 TS22 CMP69
34: PLAN JOB6 TS23 CMP69
35: PLAN JOB6 TS24 CMP69
36: PLAN JOB4 TS20 CMP65
37: PLAN JOB0 TS21 CMP68
38: PLAN JOB0 TS22 CMP68
39: PLAN JOB0 TS23 CMP68
40: PLAN JOB0 TS24 CMP68
41: PLAN JOB0 TS25 CMP69

time spent:    1.09 seconds instantiating 0 easy, 61200 hard
      action templates
               0.05 seconds reachability analysis, yielding 2916
                    facts and 61200 actions
               0.25 seconds creating final representation with
                    2880 relevant facts, 47 relevant fluents
               1.54 seconds computing LNF
               1.99 seconds building connectivity graph
               9.62 seconds searching, evaluating 43 states, to a
                    max depth of 1
              14.54 seconds total time
```

Figure 4: Solution file

in an existing IT and to transfer the planning part results as entries of the grid manager.

References

Bartak, R., and Rudova, H. 2001. Integrated modelling for planning, scheduling, and timetabling problems. In *In Proceedings of the 20 th Workshop of the UK Planning and Scheduling SIG*.

Creeger, M. 2009. Cto roundtable: Cloud computing. *Queue* 7:1:1–1:2.

Devin, F., and Le Nir, Y. 2010a. On-line timetabling software. In *Proceedings of the 8th International Conference for the Practice and Theory of Automated Timetabling (PATAT-10)*.

Devin, F., and Le Nir, Y. 2010b. Timetabling ria in action. In *Demo session of the 20th International Conference on Automated Planning & Scheduling (ICAPS-10)*.

Distefano, S.; Cunsolo, V. D.; Puliafito, A.; and Scarpa, M. 2010. Cloud@home: A new enhanced computing paradigm. In Furht, B., and Escalante, A., eds., *Handbook of Cloud Computing*. Springer US. 575–594.

Foster, I., and Kesselman, C. 2004. The grid in a nutshell. In Nabrzyski, J.; Schopf, J. M.; and Weglarz, J., eds., *Grid resource management*. Norwell, MA, USA: Kluwer Academic Publishers. 3–13.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:2003.

Frncu, M. E. 2010. Scheduling service oriented workflows inside clouds using an adaptive agent based approach. In Furht, B., and Escalante, A., eds., *Handbook of Cloud Computing*. Springer US. 159–182.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2004. Planning in pddl2.2 domains with lpg-td. *Working Notes of the 14th International Conference on Automated Planning & Scheduling (ICAPS-04)*.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning with numerical expressions in LPG. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, 667–671.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numerical state variables. *Journal of artificial intelligence research. special issue on the 3rd international planning competition* 20.

Jin, H.; Ibrahim, S.; Bell, T.; Gao, W.; Huang, D.; and Wu, S. 2010. Cloud types and services. In Furht, B., and Escalante, A., eds., *Handbook of Cloud Computing*. Springer US. 335–355.

Lin, G., and Devine, M. 2010. The role of networks in cloud computing. In Furht, B., and Escalante, A., eds., *Handbook of Cloud Computing*. Springer US. 65–82.

Ragusa, C.; Longo, F.; and Puliafito, A. 2009. Experiencing with the cloud over glite. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, 53–60. Washington, DC, USA: IEEE Computer Society.

Villegas, D.; Rodero, I.; Fong, L.; Bobroff, N.; Liu, Y.; Parashar, M.; and Sadjadi, S. M. 2010. The role of grid computing technologies in cloud computing. In Furht, B., and Escalante, A., eds., *Handbook of Cloud Computing*. Springer US. 183–218.

Zhang, Y.; Huang, G.; Liu, X.; and Mei, H. 2010. Integrating resource consumption and allocation for infrastructure resources on-demand. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, 75–82. Washington, DC, USA: IEEE Computer Society.

Zhu, J. 2010. Cloud computing technologies and applications. In Furht, B., and Escalante, A., eds., *Handbook of Cloud Computing*. Springer US. 21–45.