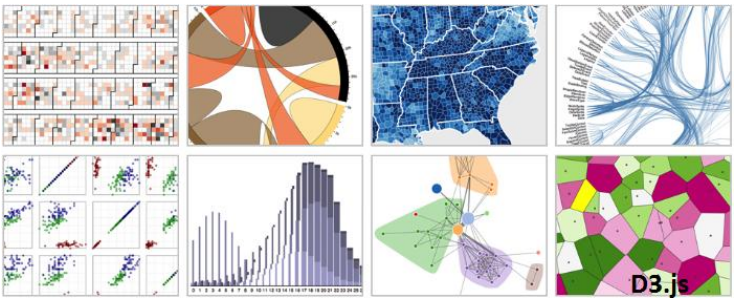
Travail d'étude et de recherche
Visualisation de l'utilisation d'une infrastructure cloud

Hedda Hedi
Engilberge Swan
Ouleha Nadir

Encadrant : Fabien Hermenier
Université de Nice Sophia-Antipolis
Juin 2013



BtrPlace

Table des matières

Chapitre 1 Introduction

Les architectures Cloud sont composées de plusieurs centaines de milliers de nœuds. Ces nœuds ont à leur tour, plusieurs propriétés et contraintes. Cela implique une masse d'informations exponentielle. Il est donc crucial de pouvoir représenter ces architectures d'une manière simple et concise afin de rapidement identifier et résoudre les problèmes.

Nous devons fournir une application web [1] qui doit visualiser de manière simple la disponibilité de machines virtuelles [2] sur les clusters [3] dans un centre de données [4]. Pour continuer nous avons besoin de définir quelques notions qui nous suivent tout au long du rapport. Une machine virtuelle est une machine fictive tournant sur une machine réelle. Un cluster est une grappe de serveur constituée au minimum de deux serveurs. Un centre de données est un site qui regroupe des équipements constituants un système d'information d'une entreprise ou d'un centre de recherche. Notre application a pour but de superviser un ensemble de centres de données, noter que nous pouvons nous adapter à n'importe quelle plateforme du moment qu'elle respecte le format du JSON [5]. Le nom de l'application est Hotplaces.

Quand nous parlons de visualiser, dans ce contexte cela signifie prendre en compte les états et placements des divers éléments mais aussi la qualité de service mais il faut aussi prendre conscience que le centre de données a une hiérarchie qui comprend plusieurs dizaines de milliers d'éléments.

Pour visualiser les données reçues par BtrPlace [6], un logiciel mis en place par notre encadrant lors de sa thèse, nous devons rendre une visualisation treemap [7], une vue hiérarchisée d'un arbre implémenté via la librairie D3 [8] (Data-driven documents) compatible avec les différentes données de BtrPlace telle que les ressources ou les contraintes d'une machine virtuelle et d'un nœud. Nous devons pouvoir zoomer mais aussi différencier l'état d'un nœud ou d'une machine virtuelle si elle ne respecte pas une catégorie de contrainte de BtrPlace.

Tout au long de ce rapport nous allons vous expliquer comment nous somme partie de rien pour faire notre application, en passant par pourquoi faire une telle application, puis nous allons voir comment le développement s'est déroulé, et enfin la gestion de projet qui a permis à ce dernier d'être aboutie.

Chapitre 2 Etat de l'art

Ce chapitre traite de l'état de l'art dans le domaine de la supervision (*monitoring*), premièrement le style de supervision à la manière de logiciel comme Nagios et ensuite de la supervision utilisant des treemap pour un aspect plus visuel qui permet de voir les règles de placement des machines virtuelles.

Le problème aujourd'hui est qu'il faut toujours être rapide dans la résolution d'allocation de ressources dans un centre de données et pour cela il faut que l'on dispose d'outils performants dans ce domaine. Il existe différents moyens pour analyser un ensemble de données, plusieurs logiciels tendent à faire cela, c'est ce en quoi consiste la supervision.

Quelques outils sont capables de donner des informations sous diverses formes. Pour citer deux outils significatifs, Ganglia [9] et Nagios [10]. Ganglia est un outil qui supervise un grand nombre de serveurs calculant la même tâches et Nagios est capable de superviser des phénomènes divers et variés tout en alertant sur des critères bien définit auparavant.

2.1 Supervision

Cette section vous présente rapidement deux outils de supervision Ganglia et Nagios et vous montre pourquoi ils ne sont pas satisfaisant à l'heure actuelle.

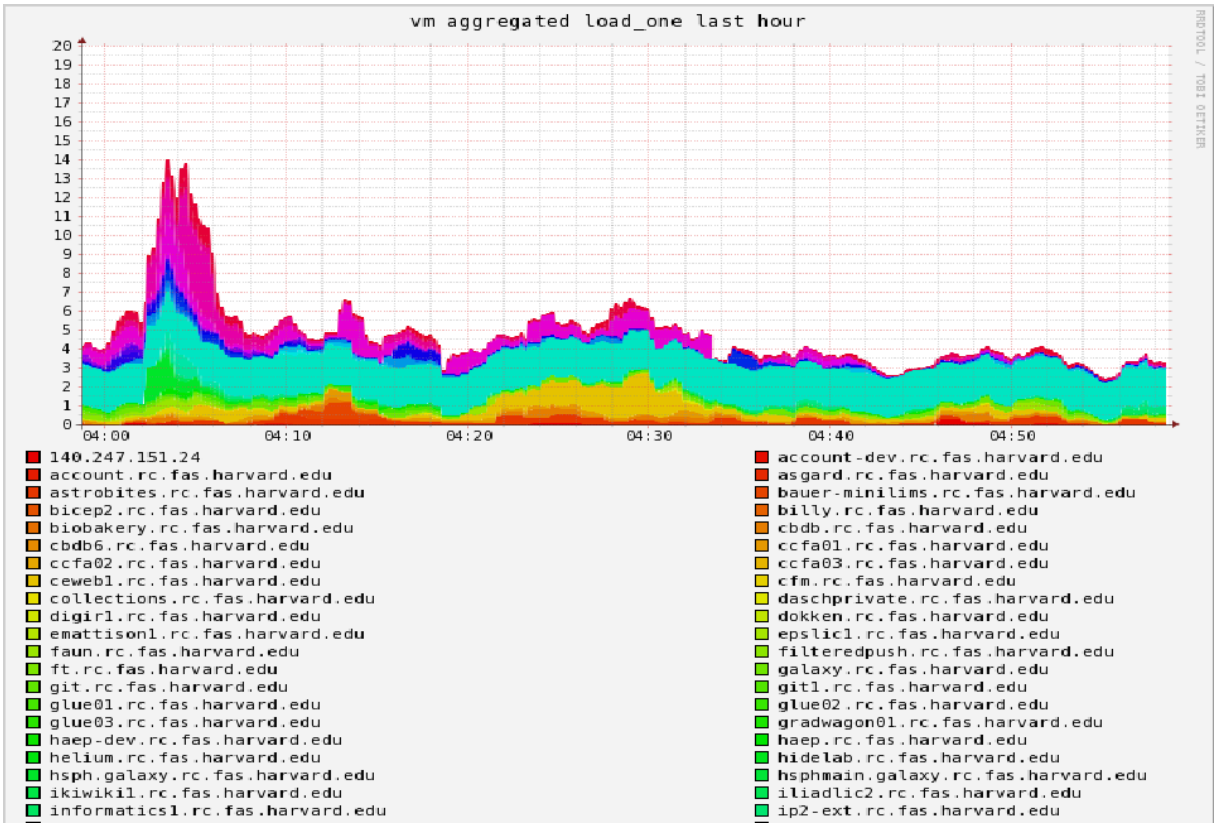


Figure 1- exemple de supervision avec Ganglia

Cette visualisation de variation ne permet pas de visualiser des règles de QoS(Qualité De Services), on ne peut pas observer les changements au niveau du placement des machines virtuelles.

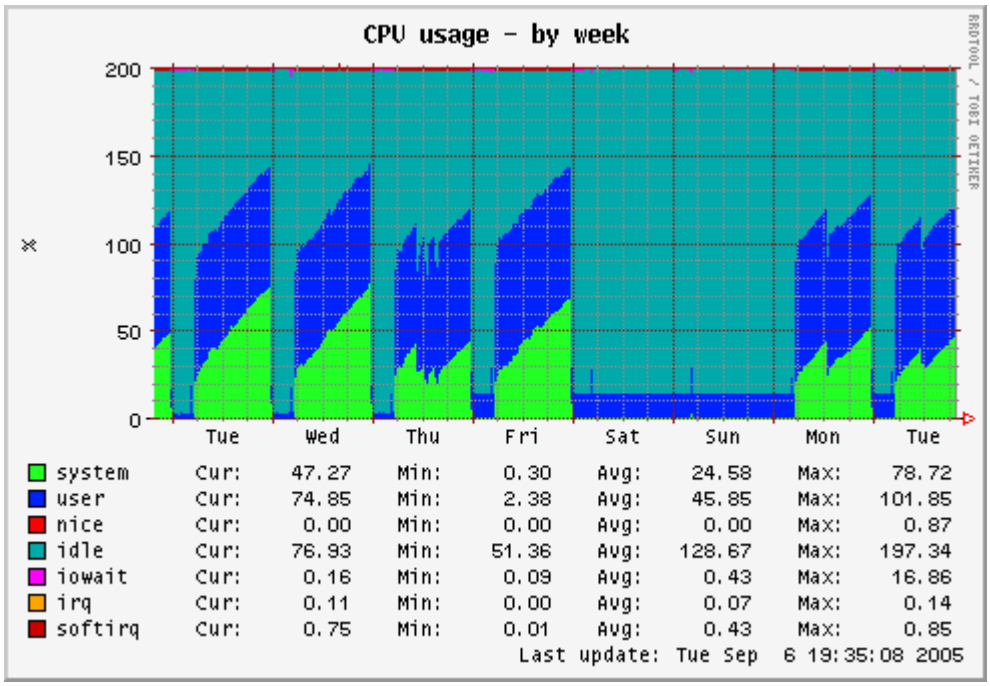


Figure 2- exemple d'interface de supervision avec Nagios

Cette image montre comment Nagios rend la visualisation pour la consommation CPU en une semaine dans un système. On ne peut pas observer les règles de placement pour les machines virtuelles, c'est une représentation uniquement orientée consommation en ressources.

Aujourd'hui, au niveau *reporting* de données, il nous faut un outil assez fluide pour avoir à la fois une vue globale des zones observées et aussi être capable en quelques secondes de visualiser la localisation d'un problème. Le plus grand point faible de ces outils est le fait que l'on ne puisse pas visualiser naturellement des problèmes de qualités de service comme par exemple les violations de règles anti-affinités, de sécurité ou encore de performances.

Les techniques qui sont le plus fréquemment utilisées sont celles qui représentent les données via une projection de plan, l'organisation en arbre ou l'organisation en réseau ou en graphe. Ces dernières étant spécialisées dans la représentation conjointe des données et de leurs relations potentielles.

2.2 Nouvelles optiques de supervision en treemap

Dans cette section nous abordons la représentation en treemap qui permet une visualisation plus fine.

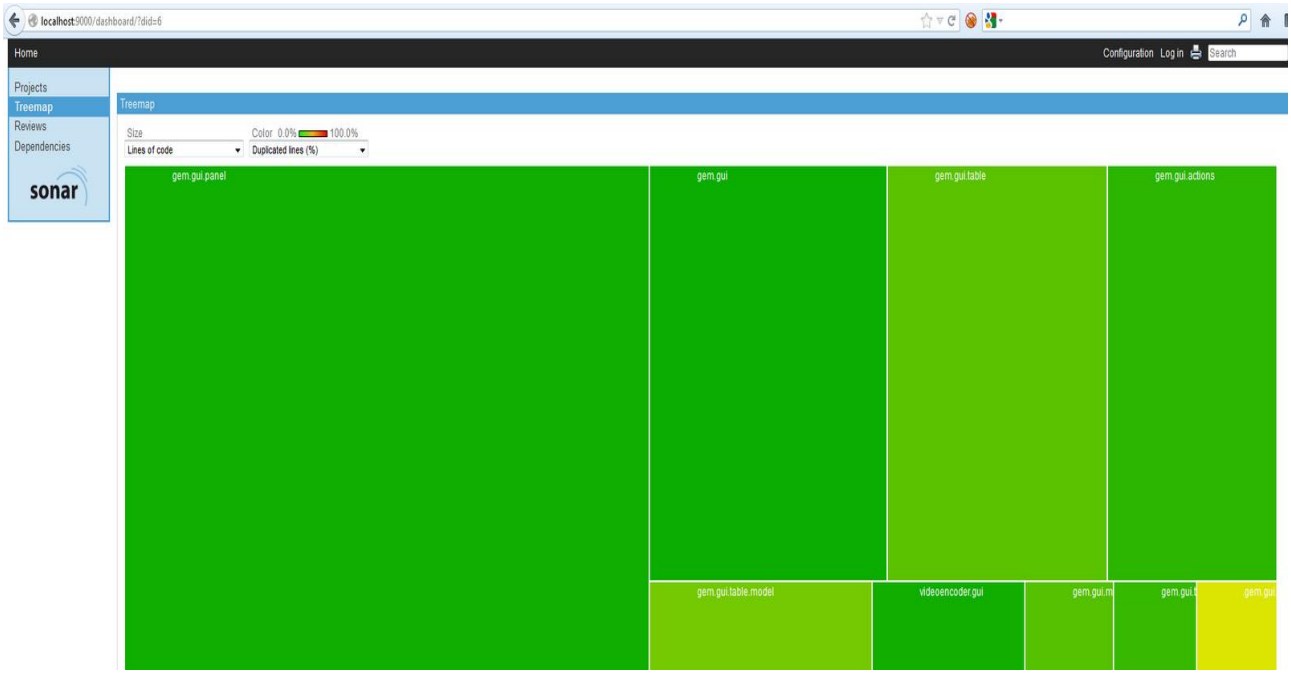


Figure 3- exemple de supervision de code avec Sonar (sous forme de treemap)

Au regard d'un logiciel tel que sonar, un outil capable de mesurer la qualité de code source sur des projets java, notre encadrant voulais quelque chose allant dans le même sens pour ce qui est de la supervision et du pilotage. En effet le *reporting* de données se fait de manière très visuelle, ce qui ne demande pas de grand effort de recherche de la part de l'utilisateur.

Pour citer un autre style d'utilisation des treemap pour visualiser des informations, il existe un utilitaire sous MAC qui s'appelle Disk Inventory X

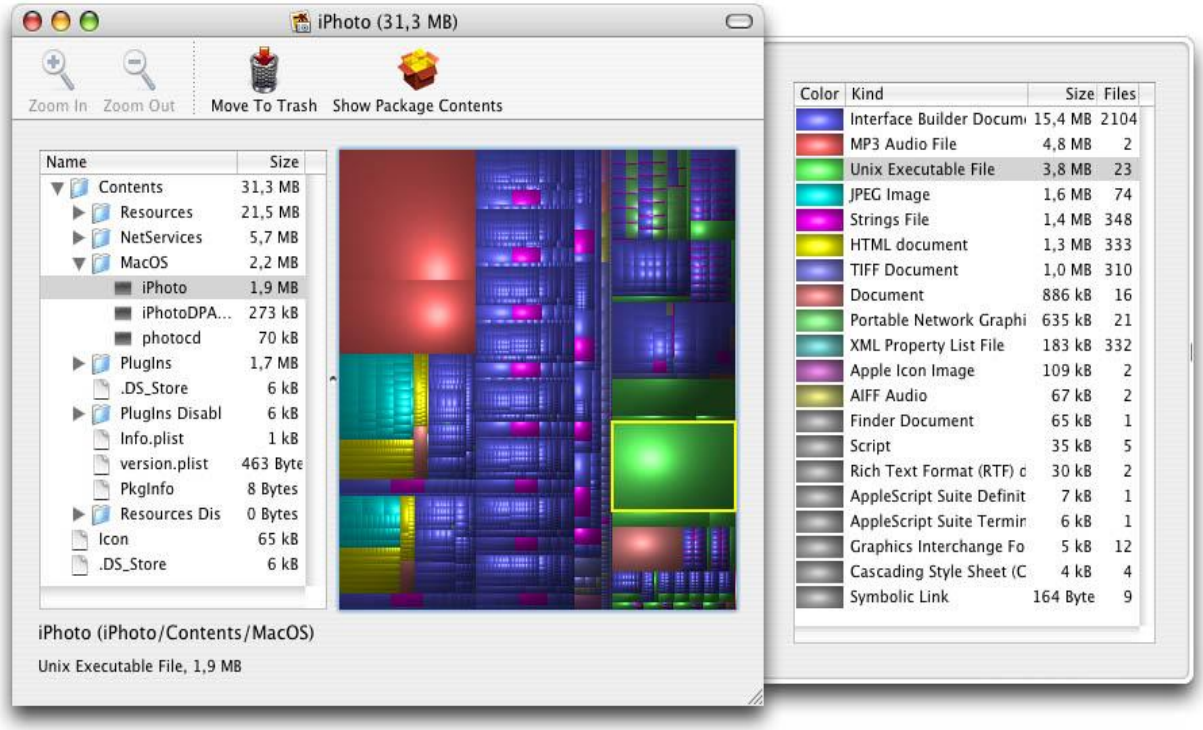


Figure 4- Interface de Disk inventory X

Encore une fois on voit tout de suite l'avantage d'une visualisation en arbre hiérarchisé (treemap), on peut voir ce qui consomme le plus d'espace disque, mais il n'y a pas d'impact direct avec le jeu de couleur, nous verrons que dans Hotplaces nous prenons en compte cela car c'est très important pour gérer des ressources afin d'avoir une vision pertinente.

C'est avec cette idée de base que nous nous somme lancé dans le développement de Hotplaces, capable de superviser avec une granularité permettant d'identifier l'erreur très rapidement.

Il est vital de pouvoir exploiter correctement les données qui existent autour de nous. Dans un cluster il en est de plus important car il faut être capable à tout moment de savoir si une ou plusieurs machines virtuelles ou même des serveurs sont en panne ou a un problème quelconque lié à une contrainte quelle qu'elle soit.

Hotplaces en sachant cela répond à cette demande tout en étant un outil performant (lié à BtrPlace), sur cette base solide nous proposons à l'utilisateur une expérience à la fois agréable mais aussi efficace pour observer l'état des machines virtuelles de son centre de données.

Chapitre 3 Présentation de Hotplaces

Dans ce chapitre nous allons aborder tout ce qui concerne la application web elle-même, c'est-à-dire de l'interface et ses choix et comment l'architecture se met en place.

3.1 Apparence :

Tout d'abord nous allons nous pencher sur l'interface de Hotplaces.

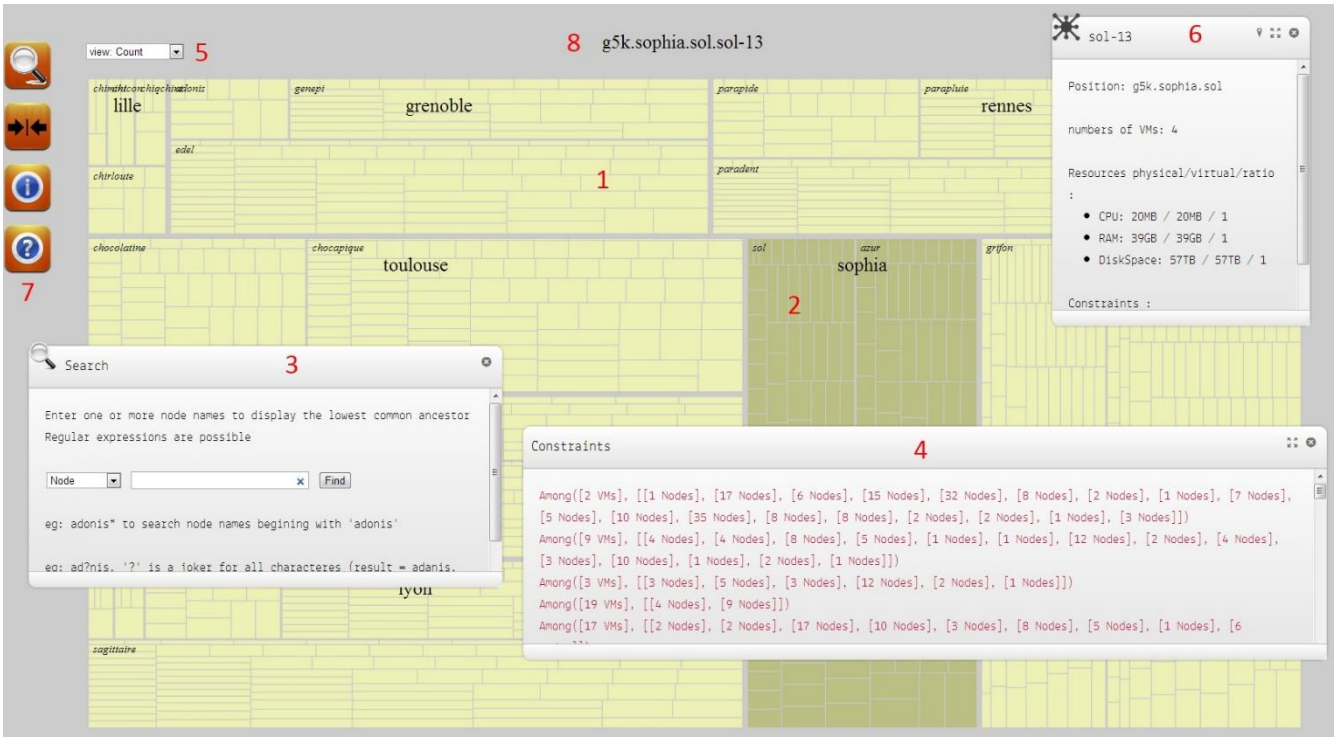


Figure 5- Interface Hotplaces

Légende :

1 : La Treemap.
2 : Zone survolée.
3 : Info-bulle pour la recherche d'un élément. (Raccourcie : 'f')
4 : Info-bulle pour les informations sur les contraintes. (Raccourcie : 'c')
5 : Liste déroulante pour réorganiser la treemap selon différentes ressources.
6 : Info-bulle contenant des informations relatives à l'élément survolé. (« espace »)
7 : Dock qui permet l'accès au différents pop-up.
8 : Correspond au chemin du nœud survolé ou la racine courante.

3.2 Architecture globale

Au niveau architectural les points importants à retenir sont les suivants :

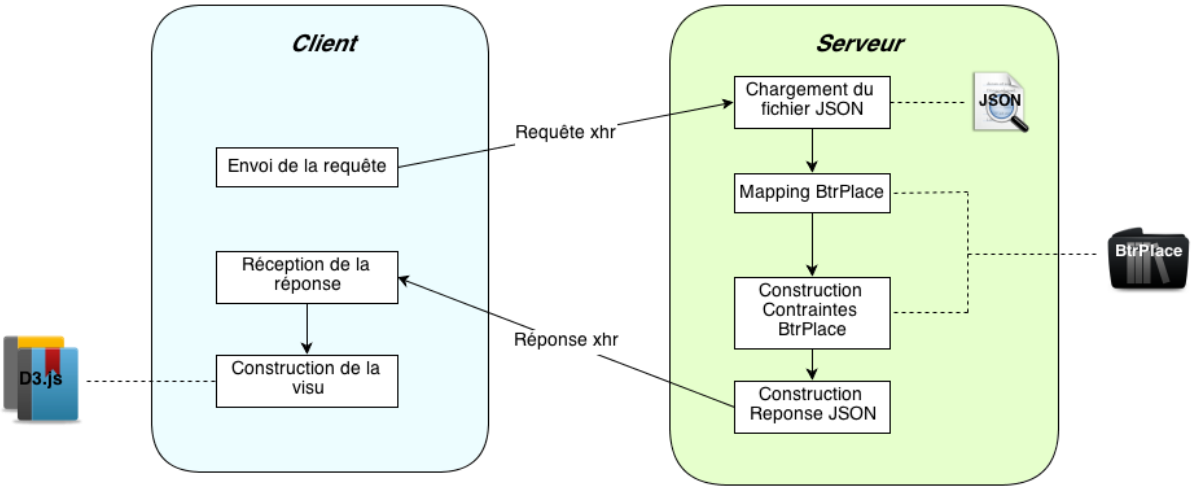


Figure 6- Architecture du projet

La communication client/serveur s'établie en REST [11] avec xmlHttpRequest [12] grâce à un serveur Jersey [13].

Le client est codé en HTML/CSS et Javascript, donc pour établir une connexion, il suffit qu'un simple navigateur web se connecte sur une adresse de la forme http://nomDuServeur:8080. Le client reste ensuite en attente de la réponse du serveur.

Le serveur quant à lui, est codé en Jersey (Java) qui permet de répondre à des requêtes REST. A la réception de la requête, il charge le fichier JSON local en mémoire, et l'analyse afin de créer un JSONObject utilisable grâce à la librairie Jettison [14].

Le fichier JSON contient toute l'infrastructure Grid'5000 au format flare, un format propre à la librairie D3 pour la représentation de données en treemap. Ainsi que la liste des contraintes liées aux nœuds et aux machines virtuelles.

Une fois le fichier JSON chargé en mémoire, le serveur parcours le fichier en créant les objets BtrPlace nécessaires pour représenter toute l'infrastructure, avec les capacités de chaque nœud, ainsi que les consommations de chaque machine virtuelle. Ensuite, le serveur parcours la partie contrainte du fichier JSON, tout en créant les contraintes BtrPlace pour les tester avec la méthode des objets SatConstraint isSatisfied() qui nous informe si une contrainte est satisfaite ou pas.

Les satisfactions des contraintes sont ensuite insérées dans le JSON réponse. Un calcul sur la surcharge des nœuds est fait par la même occasion et ajouté à chaque nœud. Et enfin quand tous ces traitements sont faits, il ne reste plus qu'à envoyer la réponse au client.

Le client reçoit finalement la réponse et procède à la construction de la visualisation de la treemap à l'aide de la librairie D3.

3.3 Gestion de l'affichage

Pour développer Hotplaces, nous avons généré, à l'aide d'un script python, une structure Json mockée, c'est-à-dire une fausse structure capable d'être interprétée par la treemap de D3 et contenant tout le nécessaire au niveau des attributs pour chaque membre de la grille. Par ailleurs, le script python génère aussi une liste de contraintes aléatoire.

Diverses technologies entre en corrélation pour l'élaboration de cette application web, en effet un serveur ainsi qu'une librairie pour représenter toute les données à exploiter sont nécessaires, de plus BrtPlace est utilisé pour récupérer les différentes visualisations qui résultent des contraintes non respectées entre machines virtuelles/clusters, nous allons en parler dans cette section.

Hiérarchie treemap et librairie D3 :

D3.js est une librairie javascript créé par Mick Bostock pour manipuler des documents basées sur des données. D3 manipule les données en utilisant le HTML, SVG et CSS. Elle est utilisable sur les navigateurs récents, combinant des composants de visualisations puissants et une approche axée sur la manipulation des données du DOM. Concrètement cette librairie nous permet de modéliser n'importe quel flux de données avec des effets de transitions fluides et agréables.

D3 est une amélioration de Protovis [15] (qui composent avant D3 des vues personnalisées avec des repères simples : bar, points, un ensemble de données…). D3 s'est imposé comme librairie de visualisation de par sa capacité à exploiter les données, par exemple il est possible de ré exploiter ces données pour dessiner un histogramme ou un autre style de graphique. Dû à sa flexibilité, D3 permet la réutilisation de code en minimisant la surcharge, D3 est très fluide au niveau de la représentation des données quel que soit leur taille.

Nous nous sommes intéressés plus particulièrement à l'implémentation via une treemap avec D3 [16]. En effet suivant l'exemple sur le site de D3, nous avons entrepris la mise en place d'une interface complète exploitant au maximum les capacités d'une treemap, avec des fonctionnalités qui comptent parmi elles le zoom, ou encore des fonctions de recherches avancées au sein de notre JSON, contenant toute l'arborescence compatible avec un centre de données du style de Grid5000 [17].

Les treemap de d3 permettent de gérer tout ce qui concerne la génération et la manipulation d'une treemap, la plus importante étant celle qui permet de dessiner la treemap : d3.layout.treemap [18].

Génération SVG :

La génération graphique de la treemap se fait à l'aide de SVG [19], en effet, chaque nœud de la treemap est représenté par un carré dessiné en SVG. Il possède une bordure et une couleur significative. La bordure permet, en changeant sa taille, de différencier les nœuds de différentes profondeurs. Par exemple, deux carrés représentant deux clusters, ont une bordure commune plus large que celle de deux carrés représentant deux nœuds. Car les clusters sont plus hiérarchiquement élevés que les nœuds.

Le Zoom :

La représentation de la treemap via SVG permet d'implémenter la fonction du zoom plus facilement. Initialement, quand la page est chargée, la structure g5k est représentée de la façon suivante : Le grand rectangle contenant toute la structure représente la racine de l'arbre g5k. Les rectangles groupés par la bordure la plus élevée décrivent les sites (Sophia, Toulouse…). La deuxième subdivision symbolise les clusters, et enfin, les plus petits rectangles visibles sont les nœuds. Pour faire afficher les machines virtuelles, l'utilisateur peut utiliser le clic gauche de la souris sur un site afin de zoomer dessus, et afficher les trois niveaux de profondeurs (glissants) suivant : sites, clusters, nœuds et machines virtuelles. Il est important de noter qu'après le zoom, les éléments de la treemap ne sont pas réagencées afin de ne pas désorienter l'utilisateur. Les zooms successif affiche naturellement le reste des éléments jusqu'aux nœuds contenant ses machines virtuelles. Pour dézoomer, il suffit de faire un clic gauche n'importe où sur la page. Il est également possible de zoomer directement vers un nœud très rapidement en double-cliquant sur le nœud en question.

Racine :

La partie haute de la treemap est réservée à l'affichage du chemin complet (depuis la racine) courant. Celle-ci ne change pas de taille après les zooms, pour ainsi garder toujours un œil sur le chemin courant. Lors du survole d'un élément avec la souris, c'est le chemin complet de ce nœud qui est afficher à la place. Il est toujours possible connaitre le nœud survolé car son nom est constamment affiché tant que la souris ne quitte pas le nœud dont il est question. Un clic sur l'un des noms constituant le chemin courant permet de zoomer vers celui-ci.

Cependant le nom des nœuds n'est pas seulement affiché dans la partie supérieure de la treemap. En effet, le nom du nœud est aussi généralement écrit sur le rectangle lui-même, comme une étiquette, pour mieux se repérer. Par contre, ces labels sont cachés lorsque le nombre de nœuds devient très grand et surtout quand la taille du carré n'est plus assez grande pour afficher le nom de façon visible. De plus l'affichage des labels ne se fait que sur les deux profondeurs plus élevés par souci d'encombrement.

Le fichier JSON :

Le fichier JSON chargé par le serveur, comme dit précédemment, suit le format « flare » propre à D3. Un fichier JSON classique est une structure basé sur des séquences de clés/valeurs. Le format « flare » restreint quelque peu l'utilisation du format JSON classique, mais en restant toutefois raisonnable. En effet, il faut savoir que D3 utilise la récursivité pour mettre en place la treemap. Et, parcours logiquement le fichier JSON récursivement. D3 exige donc que chaque nœud JSON ai un champ « name », et, si il contient des fils, d'avoir un champ-tableau nommé « children ».

Format JSON classique

```
{
   "nom": "bordeaux",
   "fils": {
      "fils1": "bordeaux1",
      "fils2": "bordeaux2"
   }
}
```

Format flare de D3

```
{
   "name": "bordeaux",
   "children": [
      "bordeaux1",
      "bordeaux2"
   ]
}
```

3.4 Recherche d'éléments

Cette section présente les différentes fonctionnalités liées à la recherche d'éléments qui sont implémenté actuellement dans Hotplaces.

Cette fonctionnalité est importante dans notre application. En effet une majeure partie de l'application s'articule autour de la recherche d'éléments dans la structure d'un centre de données.

Cette recherche consiste en une recherche classique sur des noms simples/chemins/uuid d'une machine virtuelle ou d'un nœud existant dans la structure d'arbre de données. Une recherche sur plusieurs nœuds avec différents mots-clefs est aussi possible. Dans un tel cas, les mots-clefs doivent être séparés par un espace ou une virgule.

Cette fonctionnalité utilise les expressions régulières [20], motif qui décrit un ensemble de chaînes de caractères possibles selon une syntaxe précise pour reconnaître les éléments présents dans la structure analysée.

Une autre manière de rechercher ou plutôt une méthode qui améliore la précédente consiste à utiliser ce que l'on appelle des méta-caractères [21], caractères spéciaux « * » et « ? » sont évalué dans une syntaxe similaire à celle d'un Shell Linux. Par exemple la recherche avec le mot-clef « orion* » peut retourner non seulement le nœud orion mais aussi ses fils orion-1, orion-2 etc… Pareillement avec le caractère « ? », une recherche avec le mot-clef « orion-?1 » peut retourner les nœuds orion-11, orion-21 etc…

Quand une recherche effectuée retourne plusieurs nœuds, le zoom se fait sur l'ancêtre commun le plus proche de ses nœuds, pour ainsi avoir une vue globale, sur les nœuds qui ont été sélectionnés. Une recherche sur les contraintes a été implémentée également. Un choix est fait lors de la recherche pour choisir si l'on souhaite effectuer une recherche sur des nœuds/VM ou bien sur des contraintes à l'aide d'une liste déroulante.

On peut effectuer une recherche vraiment précise dans le cas où l'utilisateur aurait que l'id du nœud ou la machine virtuelle qu'il veut observer, il peut entrer l'id dans la recherche pour trouver les informations qui l'intéresse.

Cette puissante fonctionnalité a été réutilisée en interne dans le code à plusieurs reprises, notamment dans le zoom lors d'un clic sur le chemin courant ou dans un pop-up.

3.5 Visualisation par contraintes

Cette section traite des différents moyens employés pour arriver à interpréter ce que BtrPlace nous renvoi comme alertes sur le non-respect de contraintes.

Généralités :

Toutes les contraintes ont pour but d'avoir une infrastructure soit valide, il existe différentes contraintes prise en compte, par exemple des contraintes entre éléments, c'est-à-dire entre nœuds et listes de nœuds, machines virtuelles et listes de machines virtuelles, en somme toutes les permutations de ces éléments.

BtrPlace :

BtrPlace est un algorithme de placement se basant sur des contraintes pour machines virtuelles sur plateformes d'hébergement. Une contrainte de placement est une condition qui permet de dire si un élément de la hiérarchie est à sa place. On peut personnaliser sa plateforme avec en spécifiant quelles contraintes doivent être respectées.

BtrPlace est ce avec quoi nous communiquons pour récupérer ce qu'il calcul à notre place, notamment sur le respect de contraintes diverses, par exemple « Ready » qui va décrire une ou plusieurs machines virtuelles comme « devant être dans un état Ready ».

BtrPlace est ce qui permet à Hotplaces de fonctionner correctement pour que l'on obtienne une visualisation cohérente avec la grille de données exploitée.

Intégration BrtPlace vers Hotplaces :

La partie principale du projet est celle-ci, en effet toute notre interface n'a pour autre but que d'exploiter correctement ou plutôt d'interpréter le *reporting* de données que BtrPlace nous fournis sur notre architecture de grille.

Du côté de l'interface il faut être capable de localiser où se trouve le problème que BtrPlace soulève pour nous d'une manière instantanée.

Pour cela, la structure globale de nos fichiers a changé plusieurs fois, notamment server.java et le fichier python qui génère la structure du Json, pour le rendre exploitable par BtrPlace. La tâche est d'adapter la hiérarchie suggérée par notre Json avec le mapping de BtrPlace, création de nœuds et machines virtuelles et la liaison entre eux.

Ainsi on obtient une première granularité via une première intégration de BtrPlace, qui permet d'observer les alertes, mais il est possible d'obtenir une meilleure granularité. Faute de temps, nous nous somme retrouvé dans l'impossibilité de développer d'avantage cette partie du projet, qui aurait pu être un peu plus appréciable.

Description textuelle :

Une description détaillée du nœud courant ou survolé est accessible en appuyant sur le bouton « information » ou par un raccourcie clavier grâce à la touche « espace ».
Une info-bulle apparait alors, contenant le chemin complet du nœud, la liste de ses fils (cliquables), ses ressources consommées avec son ratio, ainsi que la liste des contraintes qui lui sont liées. La couleur de ces contraintes est dépendante de la satisfaction de cette dernière.

Affichage des contraintes :

Hotplaces permet d'afficher toutes les contraintes de la structure, pour ce faire, un bouton sur le dock est dédié à cette fonction. Tout comme pour la description textuelle des nœuds, les informations sur les contraintes sont affichées dans une info-bulle. La liste est triée d'abord par satisfaction des contraintes : les contraintes non satisfaites sont placées au début, et écrites en rouge. Puis, par ordre alphabétique afin de retrouver rapidement une contrainte par son nom. En cliquant sur une contrainte, la liste des nœuds/VMs est affichée, et, comme pour une recherche multiple, l'ancêtre commun le plus proche est affiché. De plus, pour reconnaitre les nœuds sélectionnés, ces derniers clignotent régulièrement jusqu'à ce qu'une autre recherche est lancée.

Chapitre 4 Gestion de projet

Dans ce chapitre nous allons vous expliquer comment la gestion du projet s'est déroulée, premièrement un Gantt montrant la répartition des tâches (Jalons) puis avec Github comment nous avons utilisé les tickets pour attribuer les résolutions de bugs et les subdivisions en branches permettant de travailler sans perturber d'autres fonctionnalités. Enfin le calendrier du projet et les résultats obtenus mais aussi l'apport des discussions avec notre encadrant tout au long du projet.

Le TER a débutés le 14 mars avec un jeudi par semaine puis à partir du 31 mai nous nous somme entièrement consacré au TER

Nous avons convenu de partir avec la planification des tâches suivante :

Jalon 0 : Requête XHR : Envoi une requête GET qui répond OK sur la page du projet

Jalon 1 : Mock Webapp : définition minimale du design de la webapp, élaboration d'un mockJSON qui représente la structure de Grid5000.

Jalon 2 : Ressources : ajout de ressources au fichier JSON contenant le mock de grid5000. De plus, la visualisation prend en compte ces ressources en question.

Jalon 3 : Adaptation/intégration du JSON via server.java qui fait le Mapping dans un model BtrPlace, pour que celui-ci puisse mettre les contraintes non respectées. Puis de les représenter dans l'interface.

Jalon 4 : Fonctions en sus, non prévu dans le sujet ou en solution de problèmes rencontrés

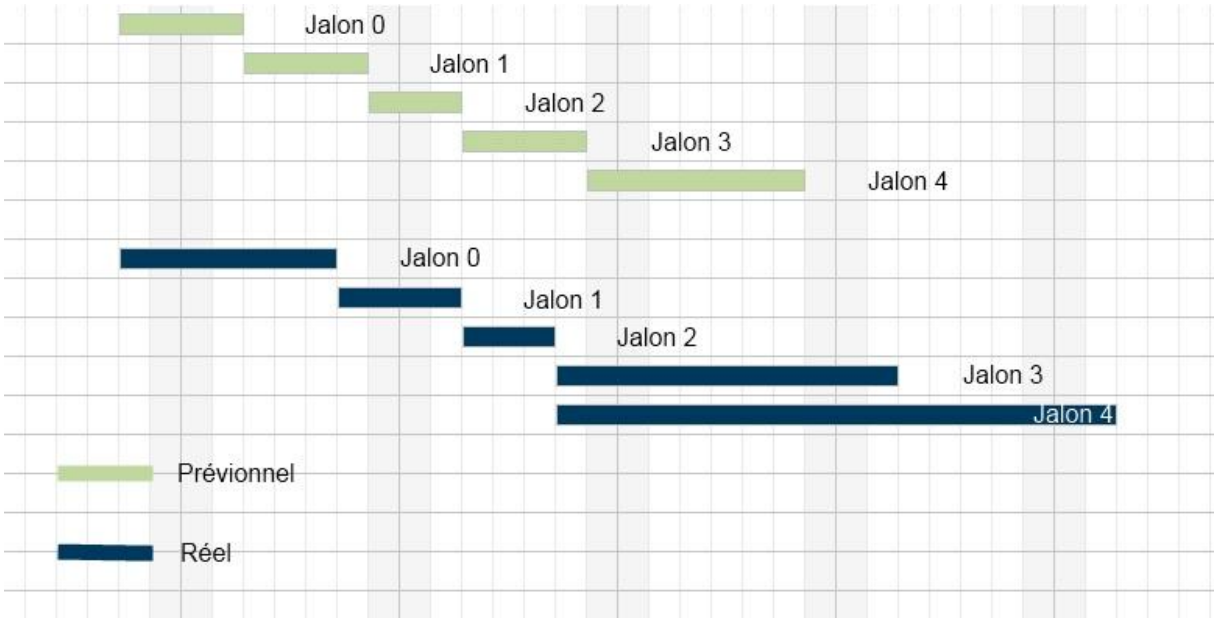


Figure 7- Diagramme de Gantt (en jours travaillées)

Le calendrier prévisionnel a été difficile à suivre à cause des contraintes de l'emploi du temps. Nous avons connu la date de soutenance que tardivement, ce qui nous a précipité dans la rédaction du rapport, aussi, certains examens ont pris le créneau horaire du TER des jeudis ce qui a été un frein dans le développement du projet. Pour finir, il y a aussi le fait que nous avions sous-estimé certaines tâches, comme la mise en place de l'interface graphique réalisée à l'aide de SVG.

L'organisation des parties importantes reste un point discutable car nous avons fait le choix de choisir nos taches selon les préférences de chacun, nous avons écarté l'orientation dite fonctionnelle.

Pour mettre en place la structure de l'environnement de travail nous avons utilisé des outils nécessaires et imposés. Tomcat [22]pour le serveur ainsi que Maven [23]comme gestionnaire de projet qui permet de déployer l'application web. Maven permet de déployé notre serveur Tomcat facilement via n'importe quelle terminal ce qui nous a permis à chacun de conserver son IDE ou éditeur de code favoris.

Le projet est actuellement hébergé sur Github, très performant pour l'organisation de code. Le fait d'avoir hébergé le projet sur Github, nous a permis d'avoir une organisation faite pour le travail en équipe, la création de tickets permet de soulever un problème à régler en l'attribuant à un membre du groupe mais encore avoir une trace de l'avancée du projet et de communiquer automatiquement à l'encadrant cette avancée et les problèmes existant.

Les branches permettent de se focaliser sur une fonctionnalité en particulier, par exemple la branche treemap qui avait pour but de dessiner la treemap à l'aide de la librairie D3, nous avions pris du retard sur cette fonctionnalité dû à l'apprentissage de quelques nouvelles technologies. Les branches facilitent également le travail collaboratif.

Effectivement nous avions commencé par développer la treemap en se reposant sur une structure HTML classique, mais on s'est aperçu après avoir obtenu le premier niveau de profondeur de la treemap, que pour zoomer dans des états plus profond, c'est-à-dire une vue différente de celle initial, où nous voyons apparaître la racine et ses fils, contenant eux même des états plus profond, nous ne trouvions pas comment conserver la forme car l'approche avec les div n'est pas graphiquement aisé [24].

Donc nous avons recherché une solution plus appropriée pour le rendu qui nous était demandé, et cette solution existe : nous avions trouvé un exemple qui faisait un zoom basique utilisant les SVG (format de données conçu pour décrire des ensembles de graphiques vectoriels et basé sur XML) de DOM, à l'aide de transition il nous était possible de définir les contraintes à respecter par le zoom, et ainsi obtenir l'effet visuel escompté.

Répartition des tâches :

Pour chaque Jalon le travail a été découpé de manière à minimiser les conflits et à ne jamais empêcher le développement des autres parties.
Le choix des parties a été fait en fonction des préférences et des compétences de chacun.

Résultats :

L'état du projet Hotplaces à la deadline est, tel qu'on peut observer, ce qui était demandé dans le sujet de TER. Nous avons une visualisation en treemap, de plus nous avons la possibilité de zoomer sur un groupe de machines virtuelles, mais aussi on peut rechercher un élément en particulier, Hotplace intègre BtrPlace pour que celui-ci puisse nous alerter de toute contraintes non respectées. Il est possible de réagencer par ressources toute la treemap, mais encore obtenir des informations sur la machine survolée en appuyant sur la touche espace. On a aussi la possibilité de représenter les ressources libres des nœuds, quand cela est possible. Des liens sont présents pour faciliter la navigation entre chaque élément (dans les bulles d'informations), cela repose naturellement sur la fonction de recherche qui nous permet d'afficher une vue sur n'importe quel élément de la hiérarchie.

Discussions avec l'encadrant :

Celles-ci nous ont permis d'évoluer dans l'avancement du projet. Lorsque l'on avait des lacunes ou des éclaircissements sur certaines fonctionnalités, monsieur Hermenier nous vient alors en aide avec quelques conseils, notamment sur des techniques pour simplifier la manière de résoudre notre problème.

Pendant la période d'une journée hebdomadaire dédiée au TER, une synthèse d'un rapport par mail est faite, détaillant l'avancement de notre projet dans cette journée. Nous discutons aussi directement avec notre encadrant lorsque cela est nécessaire, vu que la majeure partie du développement du TER se déroulait aussi à l'INRIA. Lors des semaines à pleins temps, le rythme étant un peu plus soutenu nous entretenant des meetings de cinq minutes tous les matins, en plus des discussions en direct quand nous en ressentons le besoin.

Si nous nous égarons un peu du sujet, nous somme recadré vite par notre encadrant, ce qui permettais de toujours remettre en questions la répartition des tâches et surtout l'ordre de priorité de celles-ci.

En somme nous avons pu observer de prêt comment un chercheur opère quand il doit prendre des décisions ou encore lorsqu'il organise un projet en général.

Conclusion

En choisissant ce projet nous nous demandions s'il serait intéressant, ce fut le cas car nous avons pu manipuler plusieurs technologies et langages de programmation pour les faire cohabiter dans un même projet. Nous avons eu l'occasion de côtoyer des chercheurs sur leur lieu de travail et voir au sein de l'équipe OASIS que le travail en équipe était le quotidien de chacun à l'INRIA. Ce projet a permis à chacun d'évoluer dans sa manière d'appréhender un projet en équipe, il est arrivé parfois qu'on fasse fausse route lors d'un choix de mise en place d'une fonctionnalité, cela est à mettre dans les impondérables à notre stade d'étudiant mais nous tendons à devenir bien plus efficace quand on prend en compte l'expérience accumulée sur un projet d'une telle ampleur.

Une amélioration de notre projet consiste à implémenter via BtrPlace des informations plus précises au sujet des contraintes. BtrPlace contient une partie de résolution de problèmes, c'est-à-dire qu'il propose des heuristiques capable de nous fournir quels groupe d'éléments est susceptible d'être coupable lors de la violation d'une contrainte. Dans l'état actuel du projet, nous estimons que cette tâche touchera le fichier serveur, pour implémenter la partie BtrPlace Solver afin qu'il prenne en compte ses fonctionnalités. Puis du coté javascript, développer une visualisation symbolisant les nœuds susceptible de causer les violations. Nous estimons la durée de cette tâche à une semaine.

Bibliographie

[1] «Application web,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Application_web.

[2] «Expression régulière,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Expression_rationnelle.

[3] «Métacaractère,» [En ligne]. Available: http://fr.wikipedia.org/wiki/M%C3%A9tacaract%C3%A8re.

[4] «Centre de données,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Centre_de_traitement_de_donn%C3%A9es.

[5] «JSON,» [En ligne]. Available: http://fr.wikipedia.org/wiki/JavaScript_Object_Notation.

[6] «BtrPlace,» [En ligne]. Available: http://btrp.inria.fr/.

[7] «Treemap,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Treemap.

[8] «D3,» [En ligne]. Available: http://d3js.org/.

[9] «Ganglia,» [En ligne]. Available: http://ganglia.sourceforge.net/.

[10] «Nagios,» [En ligne]. Available: http://www.nagios.org/.

[11] «REST - Wikipedia,» [En ligne]. Available: https://fr.wikipedia.org/wiki/Representational_State_Transfer.

[12] «XHR - Wikipedia,» [En ligne]. Available: http://fr.wikipedia.org/wiki/XMLHttpRequest.

[13] «Jersey,» [En ligne]. Available: https://jersey.java.net/.

[14] «Jettison,» [En ligne]. Available: http://jettison.codehaus.org/.

[15] «Protovis,» [En ligne]. Available: http://mbostock.github.io/protovis/.

[16] «Zoomable treemap,» [En ligne]. Available: http://bost.ocks.org/mike/treemap/.

[17] «Grid5000,» [En ligne]. Available: https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[18] «Wiki D3,» [En ligne]. Available: https://github.com/mbostock/d3/wiki.

[19] «SVG,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Scalable_Vector_Graphics.

[20] «Expression régulière,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Expression_rationnelle.

[21] «Métacaractère,» [En ligne]. Available: http://fr.wikipedia.org/wiki/M%C3%A9tacaract%C3%A8re.

[22] «Tomcat,» [En ligne]. Available: http://tomcat.apache.org/.

[23] «Maven,» [En ligne]. Available: http://maven.apache.org/.

[24] «Treemap Div,» [En ligne]. Available: http://bl.ocks.org/mbostock/4063582.

Table des figures :