INITIATION A LA RECHERCHE

YIELD MANAGEMENT

Adrien PAJOT, Jean-Baptiste DELON

ISEP 2014

SOMMAIRE

INTRODUCTION

Qui n'a jamais réservé un billet puis s'est rendu compte que le prix avait augmenté seulement cinq minutes après ? Coïncidence ou changement ciblé ? Il est difficile de le savoir, à première vue.

Cette gestion des prix est appelé *Yield Management*. Elle permet aux sociétés de vendre ses places (train, avion, hôtel…) au meilleur prix afin de maximiser ses recettes. Sur internet, une société a différents moyen d'appliquer ce principe : elle peut se souvenir des visites de chaque utilisateur grâce à son IP ou à ses cookies pour faire varier le prix du billet lors de ses prochaines visites. Une autre technique plus simple consiste à faire varier le prix du billet pour tout le monde selon l'heure de la journée.

Tous les 2 ou 3 mois, un article sort sur ce problème (*Les Echos, Le Figaro, Le Monde, Challenge…*). Ils sont plus ou moins détaillés et conseillent toujours à l'utilisateur de supprimer ses cookies et de changer de poste mais aucun d'entre eux ne donnent des chiffres concrets.

C'est sur ce point que nous avons voulu travailler.

I. RECUPERATION DES DONNEES

Notre cible pour le projet est la société *easyJet*. Ce choix n'est pas complètement aléatoire puisque nous pensons qu'une compagnie *low-cost* a plus de chance d'utiliser les techniques de *Yield Managment* pour augmenter son chiffre d'affaire. De plus, leur système de recherche est relativement simple comme expliqué ci-dessous.

1. Scrapping

- Imitation de l'utilisateur :

Comme beaucoup de sites de réservation, la recherche se fait en deux étapes :

- Une page contient un formulaire que l'utilisateur rempli.
- Les valeurs de ce formulaire sont envoyées en *GET* sur une page intermédiaire de chargement.
- Dans le header de la réponse de cette page on trouve un champ *Set-cookie* qui contient différents cookies en rapport avec la recherche.
- L'utilisateur est ensuite redirigé sur une nouvelle page.
- Cette page va regarder les cookies de l'utilisateur qui viennent d'être créés grâce au *Set-cookie* précédent. En fonction de ces cookies, la page va afficher les vols correspondants.

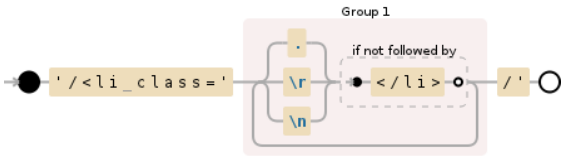
L'automatisation se faisant en PHP, il n'y a pas de gestionnaire de cookies. Le header *Set-cookie* ne sera donc pas pris en compte et aucun cookie n'est envoyé, sauf si on le précise dans une requête *cURL*.
Pour automatiser, nous nous faisons donc passer pour un utilisateur lambda en suivant les mêmes étapes que lui :

- Nous envoyons une requête sur la page de chargement. En *GET*, nous mettons les paramètres souhaités pour la recherche.
- Le *header* de la réponse de la page contient ainsi un *Set-cookie*. Nous *parsons* ce *header* pour récupérer les noms et *headers* des cookies de la réponse.
- Nous envoyons ensuite une requête à la deuxième page. Grâce à *cURL* nous plaçons dans le *header* de la requête les cookies trouvés précédemment.
- La deuxième page nous renvoie les vols correspondant à notre requête. Il ne nous reste plus qu'à la *parser*.

❖ Parsing de l'information :

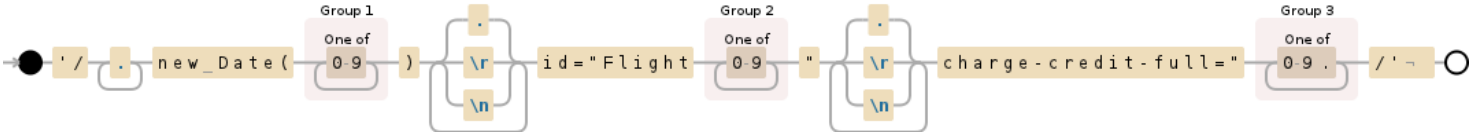
Plusieurs vols sont présent sur les pages : jusqu'à 37 différents. Nous devons donc récupérer les informations sur chacun d'entre eux. Pour faire cela, nous avons utilisé du *regex*. Chaque nouveau vol commence par un *"<li class="* nous avons pu découper les résultats en vols indépendants grâce au *regex* suivant :

```
'/<li class=\'((?:(?:.|\r|\n)(?!<\/li>))+)/'
```



Une fois les sous-groupes créés nous devons en extraire toute information possible. Pour cela, nous avons encore une fois fait appel au *regex* :

```
/.+new Date\(([0-9]+)\)(?:.|\r|\n)+id="Flight([0-9]+)"(?:.|\r|\n)+charge-credit-full="([0-9\.]+)/
```



Les groupes créés par la *regex* sont ensuite récupérés dans une variable qui contient toutes les informations sur le vol.

2. Stockage des données

❖ Que stocker ?

Maintenant que nous pouvons récupérer les données sur les vols résultant de n'importe quelle requête, se pose le problème du stockage.
Nous devons bien-sûr stocker les prix. Ces prix contiennent le numéro du vol (fournit par *easyJet*). On doit être capable de savoir comment ce prix a été obtenu.
Une table contenant toutes les méthodes utilisées pour obtenir les informations. Concrètement une méthode correspond à une adresse IP ainsi qu'à une information sur la présence de cookie ou non.
Enfin, pour savoir quelle recherche a permis de trouver ce prix, chaque prix contient l'*id* d'une recherche. Une recherche contient tous les paramètres contenus dans le formulaire de recherche.

Nous avons donc trois tables :

- ***flights*** : qui contient les recherches de vols :


```
{
    "_id": {
        "$oid": "537633ed72b6ac7020000067"
    },
    "dep": "*PA",
    "dest": "AJA",
    "dd": "23/6/2014",
    "rd": "4/7/2014",
    "apax": "4",
    "pid": "www.easyJet.com",
    "cpax": "0",
    "ipax": "0",
    "lang": "FR",
    "isOneWay": "off",
    "view": "w",
    "url":
"http://www.easyJet.com/links.mvc?dep=*PA&dest=AJA&dd=23/6/2014&rd=4/7/2014&apax=4&pid=www.easyJet.com&cpax=0
&ipax=0&lang=FR&isOneWay=off&searchFrom=SearchPod|/fr/&view=w"
}
```


- ***methodes*** : qui contient les différentes couples (IP, cookies) :


```
{
    "_id": {
        "$oid": "537633ec72b6ac7020000066"
    },
    "ip": "85.68.39.155",
    "cookies": "no"
}
```


- ***prices*** : qui contient les informations sur le vol (date, prix, retour/aller, numéro de vol) ainsi que l'*id* d'une méthode, l'*id* d'une recherche et la date à laquelle l'information a été trouvée :


```
{
    "_id": {
        "$oid": "537633f172b6ac7020000068"
    },
    "curDate": "2014-05-16 17:51:12",
    "flightDate": "2014-06-16 12:25:00",
    "flightNo": "4169476",
    "flightPrice": "91.0900",
    "idFlight": "537633ed72b6ac7020000067",
    "idMethode": "537633ec72b6ac7020000066",
    "kind": "start"
}
```

❖ Comment stocker les données ?

Ayant déjà travaillé à de nombreuses occasions avec *MySQL*, nous avions choisi ce mode de stockage au début du projet pour des questions de simplicité. Mais nous nous sommes vite rendu compte que si nous voulions stocker une grande quantité de donnée, *MySQL* ne serait pas la bonne solution. Nous avons donc fait deux choix pour gérer la quantité de données :

- Nous ne stockons des prix que lorsque ceux-ci ont changés, ce qui réduit grandement le nombre de données. Cela permet aussi de récolter plus souvent les données sans pour autant faire grossir la base de données linéairement.

- L'autre choix est d'utiliser une base de données *NoSQL* pour pouvoir stocker une grande quantité de données.

La base de données *NoSQL* choisie est *MongoDB* du fait de sa facilité d'utilisation. De plus, elle est présente sur *OpenShift* gratuitement. Nous nous sommes vite rendu compte que la base de données *MongoDB* d'*OpenShift* ne serait pas suffisante puisque qu'on en peut y accéder que depuis un site interne ; les différents serveurs n'allaient donc pas pouvoir y accéder directement et le débogage serait compliqué. Nous avons donc opté pour *MongoLab* qui offre une base de données *MongoDB* gratuite, très facile à mettre en place et accessible depuis n'importe où.

3. Serveurs

Pour avoir différentes IP et donc simuler différents utilisateurs, nous avons dû mettre en place plusieurs serveurs.
Pour cela nous avons mis en place un serveur sur *OpenShift* (gratuit) ainsi que 3 autres serveurs sur des *Raspberry Pi* en région parisienne.
Pour les mettre en place, rien de plus simple :

- Avoir PHP avec *cURL* installé ainsi que *Apache*
- Ouvrir et rediriger le port 80 sur le serveur
- Ajouter la clef SHH du serveur sur le *repo git*
- Cloner le projet sur le serveur en question dans le répertoire */var/www/*
- Récupérer l'adresse IP du serveur

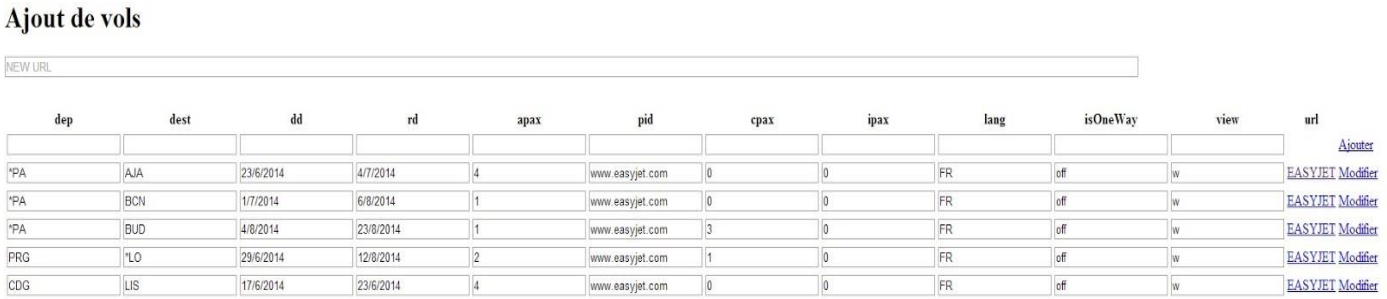
Pour automatiser la relève des prix, nous avons mis en place des tâches *Cron* sur un des serveurs :

```
*/15 * * * * wget -O - "http://IP1/easyJet/indexMongo.php?cookies=no" >/dev/null 2>&1
*/15 * * * * wget -O - "http://IP2/easyJet/indexMongo.php?cookies=no" >/dev/null 2>&1
…
```

Les serveurs sur les IP IP1, IP2… sont donc appelés toutes les 15 minutes et lancent la page qui va relever les prix. Ces prix (si différents de 15 minutes avant) sont stockés dans la base de données sur *MongoLab* avec la bonne méthode.

4. Ajouts de vols (recherches)

Pour ajouter des vols à surveiller, il faut ajouter une recherche. Ces recherches étant stockées dans la base de données, celles-ci sont récupérées et exécutées par chaque serveur toutes les 15 minutes.
Pour ajouter ces recherches, nous avons créé une interface graphique :



Il suffit d'ajouter l'URL de la requête que l'on peut trouver dans le debugger du navigateur en lançant la recherche sur le site d'*easyJet*.
Comme par exemple :

```
http://www.easyJet.com/links.mvc?dep=CDG&dest=LIS&dd=17/6/2014&rd=23/6/2014&apax=1&pid=www.easyJet.com&cpax=0
&ipax=0&lang=FR&isOneWay=off&searchFrom=SearchPod|/fr/
```

Les paramètres *GET* du lien sont extraits en JavaScript puis ajouté dans la première ligne. On peut ensuite les modifier si on le souhaite puis ajouter la recherche à la base de données. On peut aussi modifier n'importe quelle recherche de la base de données.

5. Cookies

Un de nos buts de départ était de détecter le *Yield Managment* grâce aux cookies. Nous avons donc mis en place le système de méthode qui prend en compte l'IP ainsi que la présence de cookies ou non. Cependant au moment de mettre en place la récupération des cookies nous nous sommes rendu compte qu'une grande partie de ces cookies étaient donnés par des éléments chargés en AJAX sur la page. Un simple *wget* sur la page ne nous permet pas de charger les éléments en AJAX : nous n'avons donc pas accès à de nombreux cookies.

Pour résoudre ce problème on aurait dû mettre en place un système plus évolué que le *wget* pour récupérer les pages comme *Selenium WebDriver* allié avec un navigateur. Par manque de temps et de moyens technique (nos serveurs n'ont pas d'interface graphique il est donc impossible/compliqué de lancer un navigateur) nous avons décidé de ne pas mettre en place le système de relève des cookies.

6. Résultats

Nous avons installé le projet sur 4 serveurs. Un des serveurs ayant une IP plus ou moins dynamique nous avons eu 7 méthodes différentes (7 IPs différentes).
Nous avons ajouté seulement 5 recherches : une recherche équivaut à environ 35 vols ; nous surveillons déjà plus de 200 vols. Plus de recherches n'auraient fait que complexifier les résultats. De plus, en faisant plus de 5 requêtes toutes les 15 minutes 24h/24 7j/7 les serveurs auraient pu être bannis par *easyJet*.
En 5 semaines, nous avons fait 14 000 requêtes (3 500 par serveurs) et détecté plus de 10 000 changements de prix. Grace à *MongoDB*, la base de données pèse moins de 3 Mo.

II. TRAITEMENT DES DONNEES ET AFFICHAGE

Dans cette partie, il ne s'agit pas seulement d'afficher les données, mais de les traiter au préalable.

1. NO SQL, *NO Display* (*"Not Only Display"*)

Comme nous utilisons une base de données non relationnelle – *Mongo DB* –, il s'agit, certes, de requêtes plus simples que celles des bases de données relationnelles, mais les données récupérées sont inexploitables à l'état brut.

Il est donc nécessaire de les traiter pour les rendre exploitables, mais aussi pour ne filtrer que les données intéressantes. Il ne s'agit donc là pas seulement d'un simple affichage.

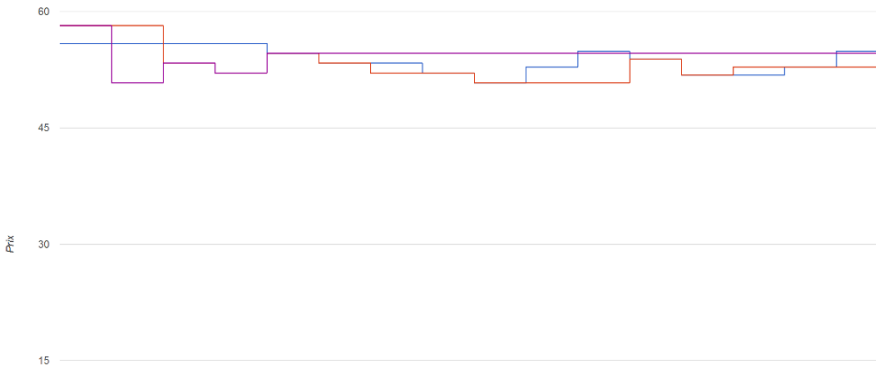
D'autres difficultés sont perceptibles comme la non-redondance des données dans la basse de données, qui impose un traitement pour détecter les persistances de données afin de recoller les intervalles des différentes séries. En effet, lorsqu'un prix reste le même au cours du temps pour un même vol, et une même méthode (adresse IP), aucun enregistrement n'a lieu afin de limiter le stockage.

Une dernière difficulté importante est la vitesse de traitement des données. En effet, côté serveur, les algorithmes se doivent d'être optimisés afin de limiter le temps de traitement PHP : pas de boucles imbriquées afin de limiter au maximum le nombre d'itérations. Comme il a de plus en plus de données, il faut prévoir que le nombre d'itération ne cesse d'augmenter. Enfin, côté client, il s'agit aussi d'optimiser astucieusement le JavaScript de sorte que les graphiques soient générés le plus rapidement possible sans surcharger les données envoyés à l'API des graphiques.

2. Librairie pour les Graphiques

Pour les graphiques, il existe de nombreuses API JavaScript permettant de les générer simplement. Certaines sont payantes, d'autres gratuites. Nous avons choisis une librairie gratuite permettant de générer simplement, rapidement et une grande diversité de graphiques : Google Charts. Nous aurions pu choisir d'autres librairies comme Flotr2, Chart.js, Nvd3, HighCharts, ou Am Charts, mais ces librairies ne sont pas toujours bien documentées, et ne proposent pas autant de type de graphiques et d'options que Google Charts ; ce qui nous aurait fait utiliser des librairies différentes en fonction des types de graphiques que nous souhaitions.

Google Developers



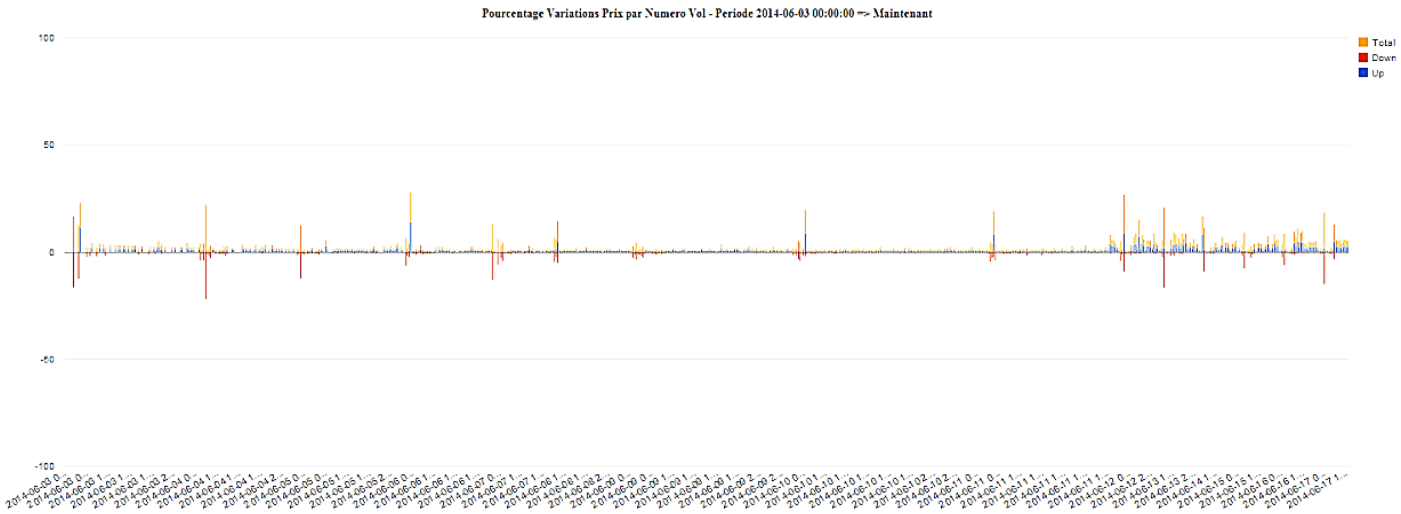
3. Les graphiques

Afficher les changements de prix en fonction de différents facteurs est primordial pour détecter ou non l'*IP Tracking*. Le *rétro-engineering* est la méthode que nous avons utilisée pour observer le *Yield Management*. Pour cela, nous devons analyser des données, et plus particulièrement les bonnes données parmi les nombreuses données que nous avons récupérées.

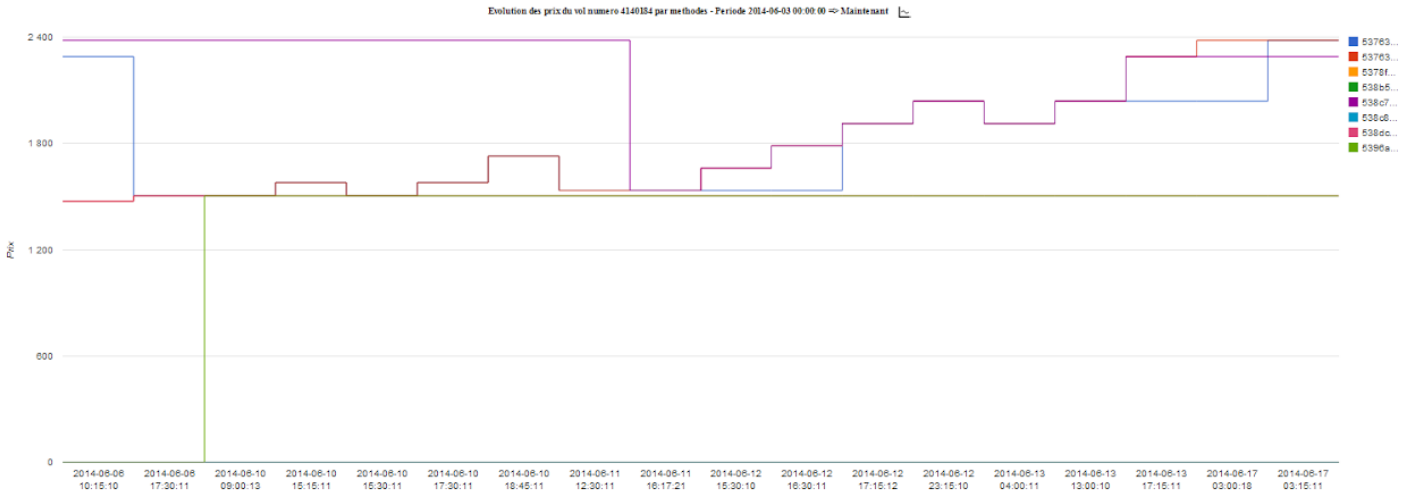
Au départ, nous n'avions pas pensé à filtrer les données. Nous souhaitions avoir toutes les données sur un même graphique, probablement, car nous n'avions pas encore capturé suffisamment de données à traiter. Nous avions réalisé un graphique qui affichait tous les prix de tous les vols (avec qu'une seule méthode) sur toute la période disponible sur la base de données. Avec les ajouts de vols et les méthodes (différentes adresses IP), ainsi que les données qui se sont accumulées, le graphique était de plus en plus long à charger, et surtout de moins en moins visible car beaucoup de données concentrées sur une petite zone. Il a donc fallu revoir nos données à exploiter.

Comme nous souhaitons voir les facteurs qui influent sur le prix des vols, il est nécessaire de prendre en compte un facteur dans au moins un graphique et de fixer les autres. Nous souhaitons visualiser les changements de prix en fonction de deux facteurs : l'**horaire** et l'**adresse IP**.

Le premier type de graphique que nous avons mis en œuvre est le nombre de changement de prix d'un ensemble de vols en fonction du temps pour une adresse IP fixée. En série, nous avons mis trois éléments : le nombre de diminutions de prix, d'augmentations de prix et total (augmentations + diminutions). Le nombre de vol augmentant au fur et à mesure du temps, nous avons souhaité représenter un pourcentage de changement de prix sur le nombre total de vol à l'instant t. Le pourcentage est plus représentatif qu'un nombre.



Le deuxième type de graphique permet de visualiser des différences de prix éventuelles entre différentes IP pour un vol fixé à un instant donné. Pour cela, nous avons placé en série les différentes IP, et nous avons créé autant de graphiques qu'il y a de numéros de vols différents ayant plusieurs adresses IP. Il a fallu faire en sorte de régler le problème de persistance des données dans le cas où le prix n'a pas changé, pour éviter les retours à zéro. Cependant charger autant de données sur plusieurs graphiques peut s'avérer fastidieux en termes de traitement. L'optimisation du code PHP et le chargement asynchrone du JavaScript permet de répondre à cette problématique.



4. Quantification des changements de prix

❖ Principe de base :

- Données d'entrée :

```
array (size=8461)
  0 =>
    array (size=5)
      'curDate' => string '2014-05-16 17:51:12' (length=19)
      'flightNo' => string '4169476' (length=7)
      'flightPrice' => string '91.0900' (length=7)
      'idFlight' => string '537633ed72b6ac7020000067' (length=24)
      'idMethode' => string '537633ec72b6ac7020000066' (length=24)
  1 => [...]
```

Il s'agit du tableau *$flightDataArray*. Celui-ci contient autant de sous tableaux que d'entrées. Chaque sous tableau est parcouru dans la boucle principal.

- Boucle principale :

Elle est scindée en deux parties : une partie incrémentation des variables pour les changements de prix, et une partie d'enregistrement des changements dans le tableau final *$diff*.

- Données en sortie :

Le PHP retourne le tableau *$diff* qui contient les données retournées par le biais de la fonction *dataToChartable()*. Le tableau PHP contient autant de sous tableaux qu'il y a d'éléments sur le graphique. Dans chaque sous-tableau, il y a un champ pour la date, et un champ pour chaque série (*up, down, total*).

```
array (size=770)
  0 =>
    array (size=4)
      'date' => string '2014-06-03 00:00:03'
(length=19)
      'up' => int 0
      'down' => int 0
      'total' => int 0
  1 => [...]
```

❖ Adresses IP :

Comme précisé ci-avant, les adresses IP (les méthodes) ont été fixées pour ce graphique. Le fichier de configuration contient la variable *$idMethodeVarMethod*. Par défaut, la valeur a été fixée à *'53763603318161cd1c8b4567'*, l'IP qui existe depuis le début des captures des données, mais cette valeur peut être changée. Fixer l'IP permet d'éviter les données redondantes : lorsqu'un changement de prix a lieu sur un vol, il a lieu pour toutes ses IP dans le cas où il y a plusieurs adresses IP sur ce vol. Cette modification a lieu dans la boucle, grâce à la condition *if($flightDataArray[$i]['idMethode'] == $idMethodeVarMethod) {*. *Voir dans la section Extrait du code : fonction getDiffArray() les appels de cette fonction.*

❖ Ignorer les vols ajoutés :

Autre difficulté : ignorer l'ajout de vols. Un ajout de vol engendre forcément un changement de prix car sa valeur initiale était nulle. Cela se fait au niveau de la vérification de la valeur précédente. Dans le cas où le vol n'existe pas dans l'index *$previous*, le vol est dit nouveau, et donc l'indice de la valeur précédente n'existe pas. Celui-ci vaut alors prendre la valeur de l'indice en cours, à savoir lui-même. Le prix est donc le même dans ce cas, ce qui permet de fuir cette contrainte. *Voir dans la section Extrait du code : fonction getDiffArray() les appels de cette fonction.*

❖ Curseur :

Il s'agit ici d'une fonctionnalité permettant de regrouper un cumul de changement de prix sur une période donnée. Plusieurs variables peuvent être configurées dans le fichier de configuration : *$inter* et *$cursorStart*. *$inter* concerne l'intervalle de cumul et *$cursorStart* correspond au début où l'on souhaite débuter le premier cumul (mettre à 0 normalement).

Dans l'exemple ci-contre, il pourrait être intéressant mettre une valeur *$cursorStart* de *a-b* (en secondes) pour commencer le cumul des données à partir du point *b*.
Enfin, une valeur de *c-b* (en secondes) pour la variable *$inter* est intéressant dans le cas où les données comprises dans cet intervalle sont trop rapprochées. Ainsi tous les *c-b secondes*, les données sont cumulées en un seul point. Ce point prendra pour date la date du dernier point enregistré comme étant dans l'intervalle. Cela évite d'avoir des résultats disparates sur des périodes trop courtes.

Dans le code, cela correspond à la fonction *checkInter()*. Celle-ci renvoie *true* si cette fonctionnalité est désactivée (*$inter=null*) ou si l'intervalle a été dépassé par la date courante (fin du cadre rouge). *Voir dans la section Extrait du code : fonction getDiffArray() les appels de cette fonction.*

❖ Intervalle d'étude :

Il est possible de choisir la période d'analyse pour ce graphique et celui ci-après. Il s'agit de la date du début de l'analyse au format *yyyy-MM-dd hh:mm:ss*. Il est possible de rentrer une date de début et de fin : *$startDate* et *$endDate*. La valeur *null* permet de désactiver le filtre sur l'intervalle. Spécifiquement pour ce graphique si la date de début d'intervalle a été fixée (différente de *null*), alors *$cursor* se fixe à la valeur de *$startDate*. La fonction *checkBorn()* permet de vérifier l'intervalle pour cette fonctionnalité.
Voir dans la section Extrait du code : fonction getDiffArray() les appels de cette fonction.

❖ Extrait du code : fonction *getDiffArray()* :

Voici la fonction utilisée pour traiter les données récupérées dans la variable globale *$flightDataArray*. Cette fonction retourne la variable *$diff* qui est un tableau PHP contenant les données des série pour chaque date. C'est la fonction *dataToChartable* qui convertit simplement le tableau PHP en un tableau JavaScript interprétable par l'API *Google Chart*.

```
function getDiffArray() {
    global $flightDataArray, $cursorStart, $inter, $idMethodeVarMethod, $startDate;
    $diff = array(array());
    $val = array();
    $previous = array();
    $val['up']=0;
    $val['down']=0;
    $j=0;
    $cursor = $cursorStart + $inter + strtotime($flightDataArray[0]['curDate']);
    if ($startDate!=0 || $startDate!=null) {
      $cursor=$cursorStart;
    }

    $nbCouple=0;

for ($i = 1; $i<count($flightDataArray)-1; $i++) {
//browse each event of flightDataArray
  if($flightDataArray[$i]['idMethode'] == $idMethodeVarMethod) {
    if(checkBorn($flightDataArray[$i]['curDate'])==true) {
      if(!isset($previous[$flightDataArray[$i]['flightNo']])) {
        $ii=$i;
        $nbCouple++;
      } else {
        $ii=$previous[$flightDataArray[$i]['flightNo']];
      }
      $previous[$flightDataArray[$i]['flightNo']]=$i;

      if ($flightDataArray[$i]['flightPrice'] != $flightDataArray[$ii]['flightPrice']) {
        //if flightPrice different from previous
        //and so if not first one because of (1 : $i=$ii)
          if ($flightDataArray[$i]['flightPrice'] > $flightDataArray[$ii]['flightPrice']) {
            $val['up']++;
          } else {
            $val['down']++;
          }
      }
      if($flightDataArray[$i]['curDate'] != $flightDataArray[$i+1]['curDate']) {
          //if next hop event in flightDataArrray
        if(checkInter($flightDataArray[$i]['curDate'], $flightDataArray[$i-1]['curDate'], $cursor)==true) {
            //[optional] date $inter var checking (2)
            $diff[$j]['date'] = $flightDataArray[$i]['curDate'];
            $diff[$j]['up'] = $val['up']/$nbCouple*100;
            $diff[$j]['down'] = -$val['down']/$nbCouple*100;
            $diff[$j]['total'] = ($val['up']+$val['down'])/$nbCouple*100;
            //save diff datas in diff array

            $val['up']=0;
            $val['down']=0;
            //reset val up and down to 0
            $j++;
            $cursor = $cursor + $inter;
            //[optional] increment cursor (2)
        }
      }
    }
  }
}
return dataToChartable($diff);
}
```

Données entrée

Curseur

Boucle Principale

Adresse IP fixé et intervalle d'étude

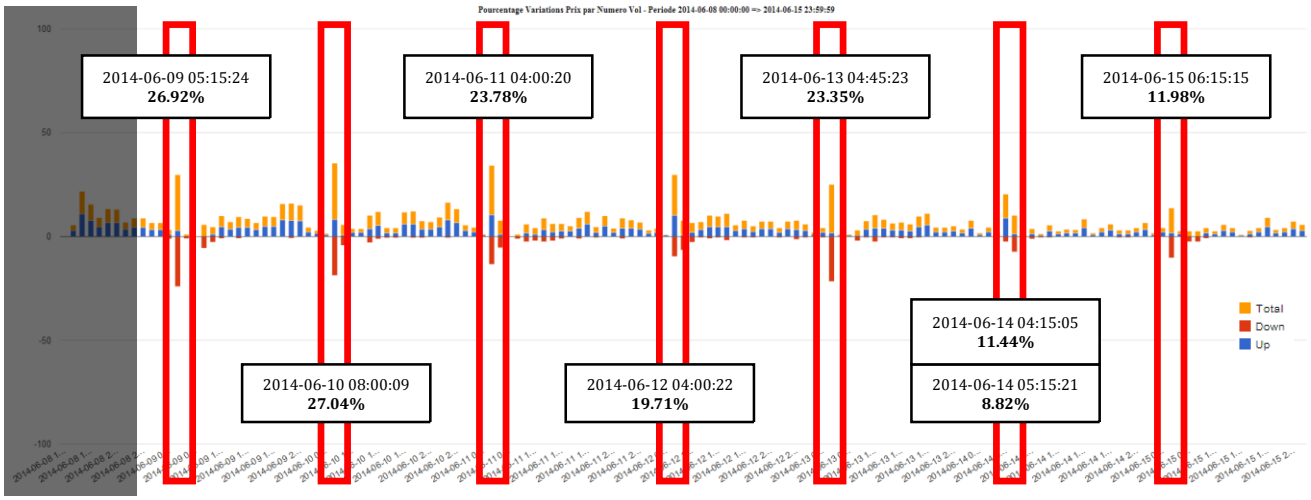
Nouveau Vol

Intervalle Incrément

Données sortie

❖ Analyse des résultats obtenus :

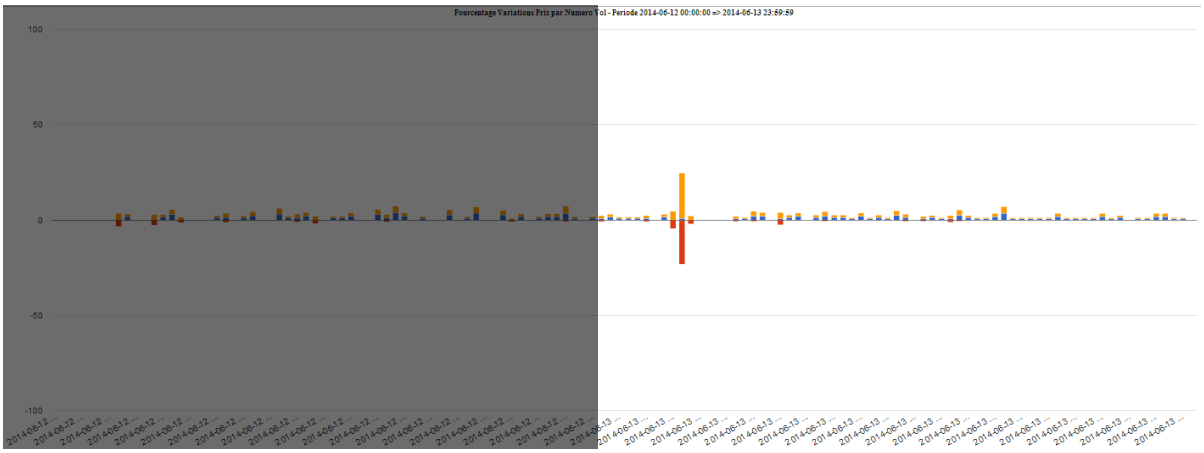
Nous avons analysé le pourcentage de variation de prix d'un vol particulier d'une semaine type : celle du lundi 9 au dimanche 15 Juin 2014. Nous avons mis un intervalle cumulatif de 3600 secondes pour cumuler les changements de prix sur une heure, et nous avons débuté l'analyse 24h avant l'intervalle, à partir du dimanche 8 Juin minuit, pour avoir les changements de la nuit du 8-9 Juin.



Les horaires peuvent varier d'une journée à l'autre. Les points ne sont pas espacés d'une heure exactement, mais d'au moins une heure. En effet, le cumul des variations peut se décaler en fonction du temps de réponse du serveur d'*easyJet* en fonction des différentes adresses IP.

On observe un cycle journalier : de nombreux changements de prix ont lieu aux alentours de 4h du matin. Certains jours, les pics oranges apparaissent plus tard (une heure après maximum) soit en raison d'un dysfonctionnement du serveur pendant cette période qui a donc reporté ces changements de prix, soit à cause d'un report du curseur à l'heure suivante car trop proche du précédent (<3600 s).
Les pics semblent indépendants du jour de la semaine et du jour du mois puisque les résultats pour d'autres semaines sont similaires.
Beaucoup d'augmentations de prix ont lieu la journée (courbes bleues), tandis que les diminutions ont lieu en majorité durant les pics matinaux (courbes rouges).

Le résultat est similaire si on se place à l'échelle de la journée, en ayant bien évidement commencé l'analyse 24h avant, de sorte à avoir le pic du matin. En restreignant cette fois si l'intervalle cumulatif à 600 secondes, on obtient une analyse identique pour la journée du vendredi 13 Juin 2014.



Cette forme de *Yield Management* semble avoir une influence importante, influence qui semble négligée par le grand public au profil de l'*IP Tracking*.

5. Différences de prix *IP-Tracking*

❖ Contrainte temps de chargement serveur :

Les données brutes en entrée sont classées par date. Or, nous souhaitons trier tout d'abord par numéro de vols, puis par méthode (série) et par date (abscisse) afin d'y enter le prix (ordonnée). Ce tri nécessite donc un certains traitement et filtrage de données qui requiert de nombreuses ressources PHP de la part du serveur. Il s'agit donc de réduire au maximum le nombre d'itérations, et de limiter les boucles imbriquées.
Au lieu d'effectuer le traitement à chaque chargement de la page, celui-ci est différé à intervalle de temps régulier dans un fichier html généré. Mais cela ne réduit que la charge du serveur, et non le temps d'exécution.
Nous avons donc optimisé le code en rangeant les données dans un tableau associatif composé *$MainArray*. Les données sont indexées par méthodes et par date, avant que le prix soit ajouté au bon numéro de vol, à la bonne IP et à la bonne date.

❖ Indexation :

① La boucle principale parcourt toutes les entrées du tableau *$flightdataArray*. Le champ *flightNo* est utilisé pour être ajouté ou ré-écrasé si déjà existant en clé principale du tableau : c'est d'ailleurs pour éviter les doublons simplement qu'il est stocké en clé.

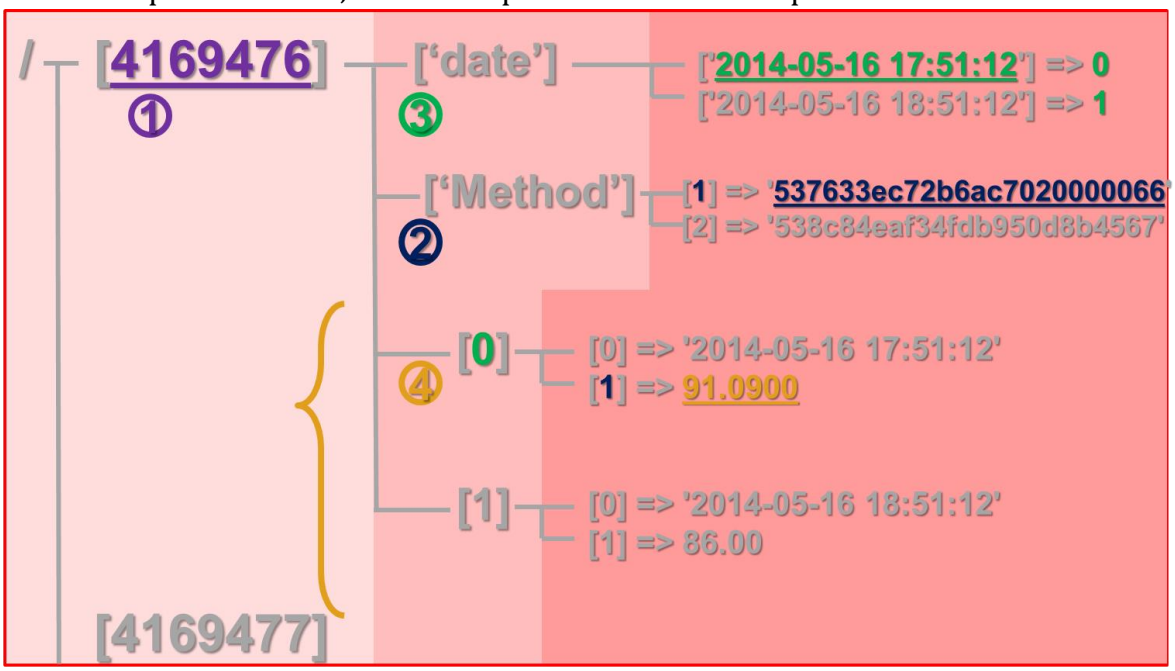
② Ensuite, le champ *curDate* du tableau *$flightDataArray* est employé pour rechercher directement dans le tableau *$flightMethod* issus des requêtes *MongoDB* la clé correspondante. A cette clé, 1 y est ajouté. Ce nouvel indice est utilisé dans le sous tableau du vol (*Method*) pour indexer l'IP à cet emplacement.

```
$flightMethod

0 => '537633ec72b6ac7020000066'
1 => '53763603318161cd1c8b4567'
2 => '5378f02572b6ac8021000000'
3 => '538b584672b6ac1c11000003'
4 => '538c7366d8291bf1238b4567'
5 => '538c84eaf34fdb950d8b4567'
6 => '538dc7edf34fdbe8078b4567'
7 => '5396acf2f34fdbdf078b4567'
```

③ Pour les dates, celles-ci sont indexées directement en tant que clé dans le sous tableau *date* du vol en question. Comme *$flightDataArray* contient, certes des données qui peuvent redonder la date, mais des données triées par date, soit la date est déjà en clé du dernier élément, soit elle n'existe pas encore : il n'y a donc pas besoin de chercher dans tout le sous tableau *date* du vol en question. Sa position dans le tableau détermine donc l'indice de la date du sous tableau.

④ Une fois ces étapes effectuées, il est alors possible d'inscrire le prix.



Lorsque le tableau *$mainArray* a été rempli convenablement. Celui-ci est parcouru une nouvelle fois pour afficher les graphiques pour chaque numéro de vol si celui-ci possède plusieurs IP : il suffit de connaitre la longueur du sous-tableau *Method*.
Afin de limiter davantage le nombre d'itération, une nouvelle fonction spécifique a été créée afin de remplacer la fonction générique utilisée précédemment pour convertir le tableau PHP en tableau JavaScript pour *Google Chart* en une fonction capable en plus de remplir les données manquantes pour les IP à un instant t. En effet, toutes les IP ne prennent pas exactement en même temps l'information (les requêtes se font une IP après l'autre). De plus, celles-ci n'enregistrent les données qu'en cas de changement de prix. A une date, toutes les méthodes liées au vol n'auront pas forcément un prix.

- Persistance des prix :

L'appel à la fonction *dataToChartableBiss()* dans l'argument de l'appel à la fonction *genereGraph()* permet de remplir les dates qui ont des méthodes vides. Dans le cas où le prix pour la méthode n'existe pas, cette fonction la créé à partir du prix pour la même méthode à la date juste avant. Cela permet de résoudre les problèmes de persistance des données en cas de non-variation du prix et dans le cas où la méthode a été ajoutée en cours de route : la valeur est alors mise à 0.
Voir dans la section Extrait du code : fonction getVarMethodArrayBis() les appels de cette fonction.

- Incrément :

Comme expliqué précédemment, les traitements sur les différents serveurs se font les uns à la suite des autres. Ainsi, il y a un léger décalage d'horaire sur les différentes méthodes et donc un décalage de l'évolution des prix. Un jeu de "*chat / souris*" charge inutilement le graphique.
Nous avons mis en place un système d'intervalle en dessous duquel deux points seront considérés comme identiques lorsque l'intervalle entre ces deux points est inférieur à l'intervalle configuré. La fonction *checkInter2()* permet d'écraser l'ancien prix par le nouveau sans créer de nouvelle date (nouveau point qui serait alors inutile) dans cette situation.
Voir dans la section Extrait du code : fonction getVarMethodArrayBis() les appels de cette fonction.

- Intervalle d'étude :

Tout comme le graphique précédent, il est possible de choisir la période d'analyse pour ce graphique grâce à la fonction *checkBorn()*. *Voir dans la section Extrait du code : fonction getVarMethodArrayBis() les appels de cette fonction.*

- Contrainte temps de chargement client :

Une fois la page générée, il s'agit de la faire afficher par le navigateur Web du client. Etant donné le nombre important de numéro de vols ayant plusieurs adresses IP (tous ont plusieurs adresses IP), le nombre de fois où le script *Google Chart* est très important. Avec la taille des tableaux JavaScript, les données sont envoyées à Google puis le script se bloque le temps que les données graphiques de Google reviennent sur la page client. A cela s'ajoute la charge de ressources de traitement par le navigateur, qui se traduit par une quantité de RAM très importante. Au-delà de 900Mo-1Go de RAM utilisée par le navigateur internet, le système d'exploitation finit par stopper le navigateur. La situation devient donc rapidement ingérable.

Nous nous sommes donc tournés vers une solution asynchrone : les graphiques ne s'affichent qu'au clique sur le titre de celui-ci. Il n'est pas nécessaire d'avoir tous les graphiques qui chargent sur la page. Il ne s'agit pas d'AJAX, puisque le traitement PHP a déjà eu lieu en un seul coup, il s'agit simplement de délestage.

Evolution des prix du vol numero 4140186 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140187 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140188 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140189 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140190 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140191 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

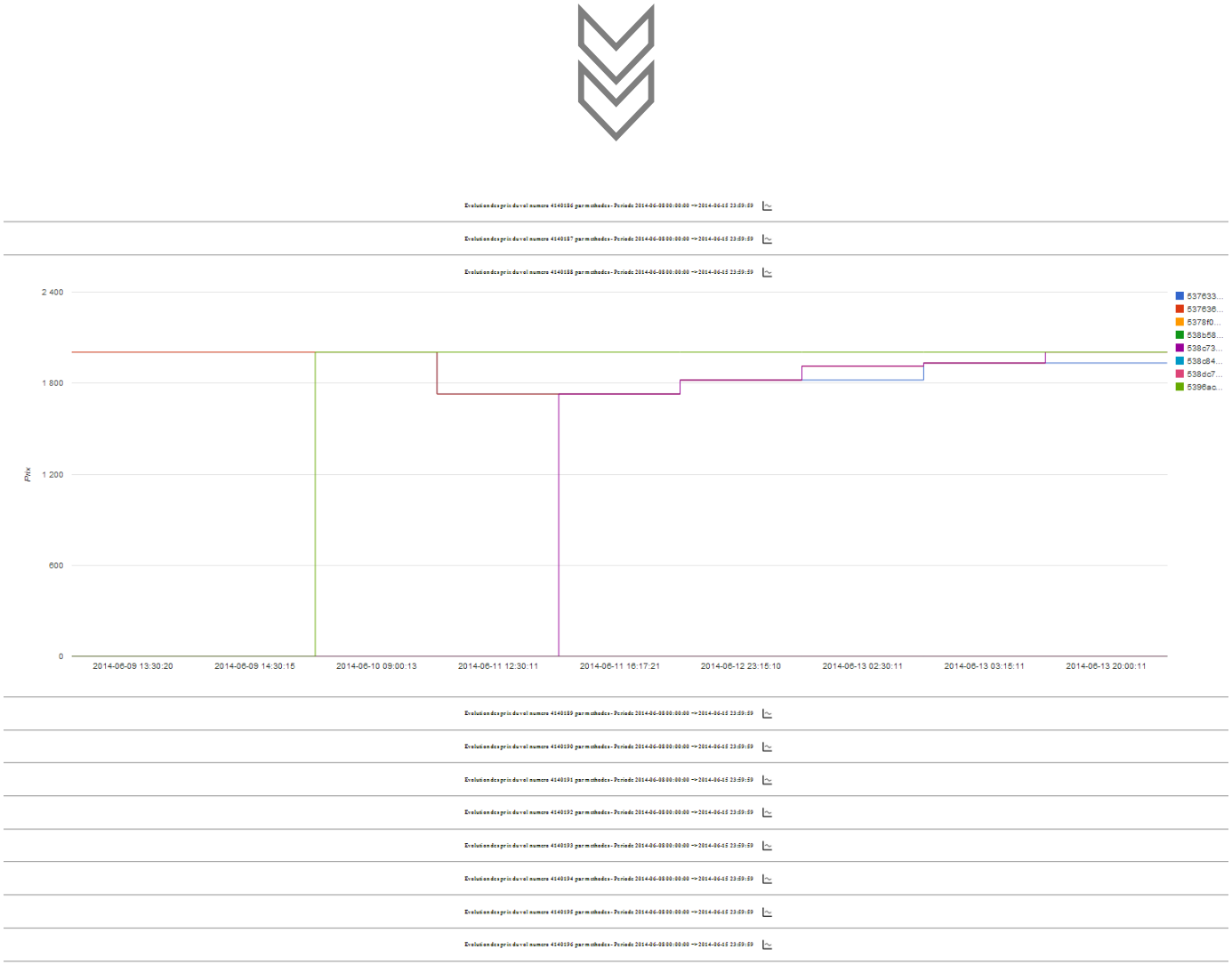
Evolution des prix du vol numero 4140192 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140193 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140194 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140195 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59

Evolution des prix du vol numero 4140196 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59



❖ Extrait du code : fonction *getVarMethodArrayBis()* :

Voici la fonction qui permet de traiter toutes les données à partir des données issues de *$flightDataArray* Afin de limiter au maximum le nombre d'itérations, et donc inévitablement les parcours de boucles, cette fonction ne retourne rien, car elle appelle la fonction *genereGraph()* qui se charge d'afficher les graphiques de chaque vol ayant plusieurs IP. La première partie concerne le remplissage de l'index *$mainArray* à partir de *$flightDataArray,* tandis que la seconde concerne la lecture de *$mainArray* pour affichage avec *genereGraph()*.

```
function getVarMethodArrayBis() {
  global $flightMethod, $flightDataArray, $flightNoArray;

  $mainArray=array(array(array()));

  for($i=0; $i<count($flightDataArray); $i++) {
    //main event loop
    if(checkBorn($flightDataArray[$i]['curDate'])==true) {
      $methodIndex=array_search($flightDataArray[$i]['idMethode'], $flightMethod)+1;
      if(!isset($mainArray[$flightDataArray[$i]['flightNo']]['method'][$methodIndex])) {
        //add index idMethod to index if not exists
      $mainArray[$flightDataArray[$i]['flightNo']]['method'][$methodIndex]=$flightDataArray[$i]['idMethode'];
      }
      if(!isset($mainArray[$flightDataArray[$i]['flightNo']]['date'][$flightDataArray[$i]['curDate']])) {
        //add current date to index if not exists
        if(!isset($mainArray[$flightDataArray[$i]['flightNo']]['date'])) {
          //initialize date index
          $dateIndex=0;
          $mainArray[$flightDataArray[$i]['flightNo']]['date'][$flightDataArray[$i]['curDate']]=$dateIndex;
        } else {
          //count date index to put next
          $dateIndex=count($mainArray[$flightDataArray[$i]['flightNo']]['date']);
          if(checkInter2($flightDataArray[$i]['curDate'], array_search($dateIndex-1,
$mainArray[$flightDataArray[$i]['flightNo']]['date']))==false) {
            //check if old date in inter or current date
            $dateIndex--;
            //will add method and price to old date without reindexing
          } else {
            $mainArray[$flightDataArray[$i]['flightNo']]['date'][$flightDataArray[$i]['curDate']]=$dateIndex;
          }
        }
      } else {
        //search current date index from date index
        $dateIndex=array_search($flightDataArray[$i]['curDate'],
$mainArray[$flightDataArray[$i]['flightNo']]['date']);
      }
      if(!isset($mainArray[$flightDataArray[$i]['flightNo']][$dateIndex])) {
        //add date header to main array if not exists
        $mainArray[$flightDataArray[$i]['flightNo']][$dateIndex][0]=$flightDataArray[$i]['curDate'];
      }
      //add price to main array
$mainArray[$flightDataArray[$i]['flightNo']][$dateIndex][$methodIndex]=$flightDataArray[$i]['flightPrice'];
    }
  }

  $nbMethod=0;
  unset($mainArray[0]);
  ksort($mainArray);
  $ii=0;
  set_time_limit(20);
  $mainArraychartable=headerToChartable($flightMethod);
  foreach($mainArray as $key => $value){
    $nbMethod=count($mainArray[$key]['method']);
    $nbDate=count($mainArray[$key]['date']);
    if($nbMethod<2) {
      //clean if less than 2 Methods
    } else {
      genereGraph($mainArraychartable, dataToChartableBiss($mainArray[$key], $nbDate), $ii, $key, 1);
    }
    $ii++;
  }
}
```

Données entrée

Boucle parcours données

Intervalle d'étude

Intervalle Incrément

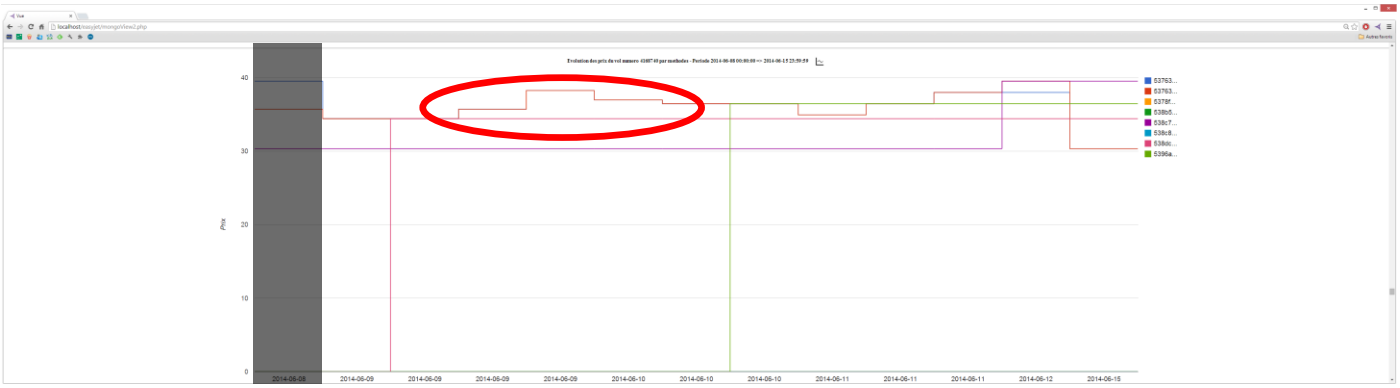
Parcours de *$mainArray*

Affichage Graphique

❖ Analyse des résultats obtenus :

Nous nous sommes focalisés sur la même semaine que précédemment, à savoir une analyse du lundi 9 minuit au dimanche 15 Juin 2014 23h59m59s, en prenant le dimanche 8 pour s'assurer d'avoir un exemple avec une variation le dimanche 8 et donc être sûr que le serveur n'était pas en panne depuis plus de 12h avant le début de la semaine d'étude.

Ci-dessous le graphique : *« Evolution des prix du vol numero 4168740 par methodes - Periode 2014-06-08 00:00:00 => 2014-06-15 23:59:59 »*, avec la date du 8 Juin 2014 grisée : *les deux dernières méthodes ne doivent pas être prises en compte* :



Même si des différences de prix sont visibles, il est très difficile d'affirmer s'il l'IP a une influence sur le prix, car s'il n'y a pas eu de changement de prix apparent, cela signifie probablement que le serveur d'*easyJet* ou le *Raspberry Pi* n'a pas pu donner de réponse, et que le changement éventuel n'a pas été enregistré dans la base de données (courbe magenta foncé). Redonder l'information dans la base de données, même en cas de non-changement de prix ou alors préciser dans un champ supplémentaire la date du dernier relevé de prix (identique au précédent ou pas) qui a fonctionné permettrait de s'assurer de la validité de l'information.

De plus, même si un intervalle de 300 secondes a été mis pour éviter les décalages de prix, il se peut que le prix ait aussi changé pendant cette période, mais qu'aucune variation n'ait été enregistrée pour une IP. Il aurait été intéressant d'enregistrer, non pas la date du serveur *easyJet* qui diffère selon le temps de latence entre les différentes requêtes des différentes IP, mais une date unique du *Raspberry Pi* maître. Cela aurait évité un post traitement fastidieux et qui peut s'avérer inexacte dans le cas énoncé ci-avant.

CONCLUSION

Ce présent rapport permet d'affirmer avec certitude l'utilisation du *Yield Management* comme stratégie commerciale des compagnies aériennes. Cette étude le montre pour *easyJet* ; nous aurions pu étendre notre étude à d'autres compagnies aériennes voire même d'autres secteurs comme celui du ferroviaire. Cependant, la mise en place et l'utilisation des APIs pour plusieurs compagnies aurait été fastidieuse et n'aurait pas apportée plus de résultats pertinents.

Les paramètres employés pour les deux graphiques sont intéressants, mais il ne concernait pas de vols long-courriers. Le continent, les fuseaux horaires sont des paramètres qui peuvent aussi s'avérer pertinents, mais moins que ceux que nous avons employés dans cette étude.

Enfin, ce rapport permet surtout de mettre en évidence l'influence de l'horaire sur le prix de l'avion. Nous n'avons pas pu le faire sur les cookies car cela s'avère compliqué dû au fait de l'API. Pour l'IP, cela reste difficile dans la mesure où le prix augmente en fonction de la journée et qu'il est impossible de synchroniser exactement à la seconde près les requêtes des divers IP vers le serveur d'*easyJet*, même avec les suggestions énoncées dans l'analyse des résultats du graphique sur l'IP Tracking.

BIBLIOGRAPHIE

1. Site d'*easyJet*

- Site de réservation : http://www.*easyJet*.com/

2. Articles à ce sujet

- Big Data : http://www.lesechos.fr/idees-debats/cercle/cercle-99291-le-cote-obscur-de-big-data-1009434.php
- Yield Management : http://sosconso.blog.lemonde.fr/2013/01/23/pourquoi-les-prix-des-trains-ou-des-avions-varient-dune-minute-a-lautre/
- Yield Management : http://www.lefigaro.fr/secteur/high-tech/2014/01/27/01007-20140127ARTFIG00552-le-prix-des-billets-d-avion-varient-sur-internet-mais-pas-selon-votre-adresse-ip.php
- IP Tracking : http://defense-du-consommateur.comprendrechoisir.com/astuce/voir/249228/dejouer-l-ip-tracking-pour-ne-pas-payer-son-billet-plus-cher
- IP Tracking : http://www.challenges.fr/entreprise/20130612.CHA0739/billet-d-avions-comment-echapper-a-l-ip-tracking-des-compagnies-aeriennes.html

3. Outils scrapping

- PHP Simple HTML DOM Parser : http://simplehtmldom.sourceforge.net/
- Selenium : http://docs.seleniumhq.org/projects/webdriver/

4. Librairies pour Graphiques

- Flotr2 : http://humblesoftware.com/flotr2/#!basic-bars
- Chart.js : http://www.chartjs.org/docs/
- Nvd3 : http://nvd3.org/examples/multiBar.html
- HighCharts : http://www.highcharts.com/demo/column-basic/
- Am Charts : http://www.amcharts.com/demos/multiple-data-sets/#theme-none
- Google Charts : https://google-developers.appspot.com/chart/interactive/docs/gallery
- Google Chart - Stepped Area : https://google-developers.appspot.com/chart/interactive/docs/gallery/steppedareachart
- Google Chart – Column Chart : https://google-developers.appspot.com/chart/interactive/docs/gallery/columnchart

5. Autres librairies JavaScript

- JQuery : http://code.jquery.com/jquery.min.js
- JQuery.scrollTo : http://balupton.github.io/jquery-scrollto/
- JSApi : https://www.google.com/jsapi