Big Data :
Initiation aux techniques de passage à l'échelle

BRIE PICARDIE

Lille 1 - Master Ingénierie Statistique et Numérique

MDC
STRATÉGIE ET PILOTAGE
DE LA RELATION CLIENT

Programme

Introduction :

Présentation des enjeux et de l'activité de la data dans l'entreprise

Première partie :

Limites du modèle traditionnel (client/serveur) et introduction des concepts de passage à l'échelle

Seconde partie :

Le passage à l'échelle en pratique

Votre intervenant

Pierre.THOREL@ca-briepicardie.fr

Formation

TELECOM Bretagne

Ecole d'ingénieur des technologies de l'information

Université Lille1 Sciences et Technologies

Master ISN
Promotion 2010

cfpb

Institut Technique de Banque
Promotion 2015

Parcours professionnel

BRIE PICARDIE

Responsable data science et études stratégiques
Depuis 2015

NORD DE FRANCE

Chef de projet MOE – Organisation de la distribution et optimisation des portefeuilles client
2014

Chef de projet MOA - Migration informatique vers un nouveau système d'informations : domaines marketing, distribution et Data
2012/2013

Expert CRM et Data
2010/2011

La data au Crédit Agricole => Les enjeux

Dans un monde où les données s'ouvrent :

- Etre capable de tirer la valeur client et marché de nos données **avant les fin-techs**, **lutter contre la fragmentation de la relation client**

- **Accélérer le développement et la fidélisation** sur l'ensemble des marchés, dans un contexte de saturation et de banalisation des sollicitations clients ➔ DVT FDC, PNB, IRC

- **Améliorer la productivité et l'efficacité de la distribution multicanale** : le bon motif de Contact, au bon Client, par le bon Canal et avec la bonne Chronologie ➔ IR Clients et Collaborateurs, réduction des coûts d'exploitation

- **Anticiper les risques** (compliance, contreparties, fraudes….) pour réduire les impacts ➔ RBE

- **Eclairer les prises de décisions** dans un **modèle bancaire en pleine transformation**

- **Limiter la dépendance** technologique externe sur les outils stratégiques, induisant une risque de **prise en otage de la valeur** (acquisition d'audience web et mobile, automatisation de processus de relation client / d'octroi / …)

Composition de l'équipe Data Brie Picardie :

Une diversité de compétences

Pierre

Responsable

Estelle

Lydie

Claire

3 Data Scientists

Igor

1 Data Engineer

Savoir-faire :

- Manipuler et fouiller de **très grandes quantités** de données
- Concevoir des **algorithmes innovants.**
- **Machine Learning / Deep Learning**

- **Industrialisation** des flux de données, **de la source au livrable opérationnel**
- Economies d'échelle / Rationalisation des ressources IT
- **Maîtrise des risques** opérationnels liés à la sécurité SI
- Culture juridique et éthique

Formations :

De quoi parle-t-on ?

Définition

Les ***big data***, littéralement les « grosses données », ou **mégadonnées** (recommandé), parfois appelées **données massives**, désignent des ensembles de données qui deviennent tellement volumineux qu'ils en deviennent difficiles à travailler avec des outils classiques de gestion de base de données ou de gestion de l'information.

Wikipédia

En pratique

La maîtrise de technologies orientées Big Data devient utile lorsqu'une base de données :

- exploite de très nombreuses variables : banque/assurance, administrations fiscales et sociales, …
- traite des données par nature volumineuses telles que la vidéo, le son : services de sécurité, YouTube, marchands numériques (stores iTunes et Windows, Google Play,), …
- concerne un nombre d'individus (clients, pages, articles, publications, flux financiers, communications) en centaines de millions ou plus : Google, Amazon, Facebook, banques, télécoms, …
- Lorsque le besoin d'assurer une très haute disponibilité des données et des traitements apparaît (CRM temps réel, défense, aéronautique, …).

Première partie

Limites du modèle traditionnel et introduction des concepts de passage à l'échelle

BRIE PICARDIE

MDC
STRATÉGIE ET PILOTAGE
DE LA RELATION CLIENT

Programme / Première partie

Le modèle client/serveur et ses limites

Des défis technologiques à résoudre pour passer la frontière du big data

- Préalable : quelques notions d'algorithmique
- Principe des architectures « big data »
- Organisation des données

Questions / réponses

La problématique du volume : exemple

DWH des Caisses Régionales (CR) **de Crédit Agricole :**
Environ 10.000 données, **avec en moyenne 1 an d'historique,** concernant les produits, les clients, les entretiens, les flux monétaires, … pour un total de 5 MO de données par client en moyenne.

CR Brie Picardie = 1M de clients soit **5 TO de données**

Les 39 CR = 20M de clients soit **100 TO de données**

Limite de SAS pour une régression, une ACP, …

Environ 30.000 individus pour 1.000 variables

La solution client/serveur classique

Stockage par baie NAS (permet de cumuler plusieurs disques sous la forme d'un seul disque logique, accès en réseau).

Le stockage des données est potentiellement illimité

Mais pas la capacité de traitement, limitée par :

- Le réseau reliant la baie aux serveurs
- Puissance de calcul du serveur SQL
- Puissance de calcul du serveur SAS

Serveur SQL
(Oracle, SQL Server)

Serveurs spécifiques
(SAS, BO, …)

Postes clients
(utilisateurs)

Ce modèle met en parallèle la capacité de stockage, mais la capacité de traitement reste en série !

Programme / Première partie

Le modèle client/serveur et ses limites

Des défis technologiques à résoudre pour passer la frontière du big data

- Préalable : quelques notions d'algorithmique
- Principe des architectures « big data »
- Organisation des données

Questions / réponses

Algorithmique

Un algorithme est une suite finie d'opérations déterministes permettant de résoudre un problème.

La performance d'un algorithme ayant pour objet de réaliser un traitement sur un ensemble de données de Cardinal n se mesure en exprimant le nombre d'opérations élémentaires nécessaires par une fonction de n. En général, on se contente d'une majoration : O(n), O(n²), … . On parle alors de complexité d'un algorithme.

Lorsqu'on manipule des volumes de données importants, ou dont le volume augmente régulièrement, il est important de maîtriser la complexité des algorithmes utilisés.

- Il existe parfois différents algorithmes de complexités différentes permettant d'atteindre un même résultat. **On recherchera toujours la complexité minimale.**
- **Sont généralement compatibles à un usage de la totalité des données les complexités ne dépassant pas O(n.log(n)).**
- **En modélisation, un échantillonnage est généralement nécessaire à partir d'une complexité O(n²).**

Exemple : Algorithmes de tri

Objet : Trier un tableau **T** de **n** éléments d'un ensemble **E** sur lequel on dispose d'une relation d'ordre **<**.

Algorithme de tri par sélection :

Initialisation : i=1

Itération : On pose s = i. Pour k allant de i+1 à n, si Tk < Ts alors s:=k. Enfin, on intervertit Ti et Ts, puis on incrémente i. (On recherche le plus petit élément parmi Ti…Tn puis on le place à la position i).

Condition d'arrêt : i=n.

Analyse de la complexité :

- Il faut réaliser n-1 itérations, chacune demandant n-i comparaisons, n-i écritures pour incrémenter k, un maximum de n-i écritures pour mettre à jour s, et 3 écritures pour initialiser s, incrémenter i et intervertir Ti et Ts.
- Soit un maximum de 3.(n-1)(n-2)/2 + 3(n-1) opérations.
- **Complexité = $O(n^2)$.**

Exemple : Algorithmes de tri

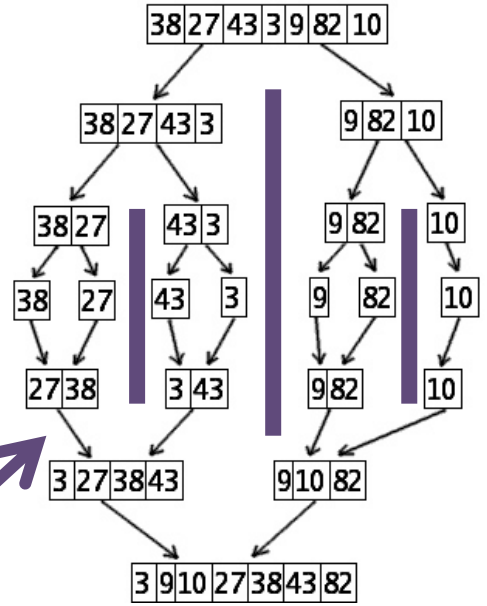
Objet : Trier un tableau **T** de **n** éléments d'un ensemble **E** sur lequel on dispose d'une relation d'ordre **<**.

Algorithme de tri par fusion :

On divise le tableau en 2 de façon récursive jusqu'à ne disposer plus que de sous-tableaux triés. (Un tableau à 1 élément est trié par définition…).

Puis, on re-fusionne récursivement ces sous-tableaux en les maintenant triés.

Complexité = O(n.log(n)).



$K = \lceil \log_2(n) \rceil$

K= Nombre d'opérations de divisions
= Nombre d'opérations de fusions

Chaque fusion s'effectue en maintenant l'ordre, en comparant à chaque fois les 2 éléments les plus à gauche pas encore sélectionnés.
Le nombre de comparaisons à faire est limité par n pour chaque niveau.

De plus, cet algorithme est parallélisable !

Illustration de l'importance d'un bon choix algorithmique

Tests réalisés in-memory, sur un système 4 cœurs à 3.5 GHz, sur un tableau de n entiers.

	Tri fusion parallèle	Tri fusion simple	Tri sélection
N = 10 000	133.616 opérations **0 ms**	133.616 opérations **0 ms**	49.995.000 opérations **109 ms**
N = 100 000	1.668.928 opérations **16 ms**	1.668.928 opérations **31 ms**	704.982.704 opérations **10 s**
N = 1 000 000 **Nb. de clients d'une caisse régionale**	19.951.424 opérations **109 ms**	19.951.424 opérations **250 ms**	$\sim 10^{11}$ opérations **16 minutes**
N = 20 000 000 **Nb. de clients de l'ensemble des CR**	486.445.568 opérations **1,0 s**	486.445.568 opérations **2,7 s**	$\sim 4.10^{13}$ opérations **5 jours**
N = 1 Mds Nb. de flux annuels d'une caisse régionale	$\sim 2.10^{11}$ opérations **15 minutes**	$\sim 2.10^{11}$ opérations **40 minutes**	$\sim 4.10^{17}$ opérations **14 ans**
N = 20 Mds Nb. de flux annuels de l'ensemble des CR	$\sim 10^{13}$ opérations **12 heures**	$\sim 10^{13}$ opérations **32 heures**	$\sim 2.10^{20}$ opérations **5.600 ans**

Extrapolation : volume ingérable in-memory pour la machine de test ou durée de la simulation non raisonnable

Principe des architectures « Big Data »

Au lieu de répartir uniquement les données sur plusieurs disques en conservant une unité de traitement unique (modèle client / serveur), on répartit également les traitements.

- Les algorithmes sont conçus de manière à paralléliser les traitements en exploitant simultanément la puissance de calcul de toutes les unités disponibles
- Si le nombre d'unités installées est lui aussi proportionnel à n, on gagne potentiellement un degré : Temps O(n) pour une complexité algorithmique $O(n^2)$.
- Si un nombre d'opérations O(f(n)) est distribué équitablement sur p unités, la complexité temps est alors de l'ordre de O(f(n)/p))

Ces modèles, adaptables à des volumes de données potentiellement illimités, sont dits distribués ou encore parallèles.

Exemple de modèle distribué : Map – Reduce.

Principe : Pour chaque traitement, on définit une fonction Map qui divise la tâche à réaliser sur les différentes unités concernées, et une fonction Reduce qui compile les résultats.

Map - Reduce

Soit E un ensemble de données, réparties sur p unités, $E = \bigcup_{i=1}^{p} E_i$

Traitements « simples » :

- Rechercher les individus vérifiant un critère donné, évaluer une fonction f définie sur E (somme, moyenne, …) pour un ou plusieurs individus, effectuer une combinaison linéaire, …

Exemple : Moyenne

$$f : E \to \mathbb{R}, \quad \underbrace{\frac{1}{\#E}\sum_{x\in E} f(x)}_{\text{O(\#E)}} = \frac{1}{\#E}\sum_{i\in\{1,\dots,p\}}\sum_{x\in E_i} f(x)$$

REDUCE : O(p)

MAP : O(#E/p)

Traitements « complexes » : Appliquer un traitement à E dont le résultat n'est pas déductible du même traitement appliqué aux sous-ensembles E1,…,Ep.

Exemple : les calculs matriciels (et donc : la plupart des modèles de régression et de classification !)

Exemple : Calcul de médiane en Map - Reduce

Soit E un ensemble de données, réparties sur p unités, $E = \bigcup_{i=1}^{p} E_i$, et $f : E \to \mathbb{R}$

Exemple d'algorithme Map Reduce pour calculer la médiane m de $\{f(x), x \in E\}$

Première estimation :

- Map() : Demander à chaque unité le calcul de la médiane m_k de $\{f(x), x \in E_k\}$. $\mathbf{O}(\lceil \log_2(n/p) \rceil . n/p)$
- Reduce() : Constituer l'ensemble $\{m_k, 1 \leq k \leq p\}$ et le trier. $\mathbf{O}(p . \lceil \log_2(p) \rceil)$

Encadrement fin de la médiane :

- Map() : $\forall i, j \in \{1, \dots, p\}^2, N_{i0} = \#\{x \in E_i / f(x) < m_1\}, N_{ip} = \#\{x \in E_i / f(x) \geq m_p\}$, $N_{ij} = \#\{x \in E_i / m_j \leq f(x) < m_{j+1}\}$. $\mathbf{O}(\lceil \log_2(p) \rceil . n/p)$

Exercice : Ecrire un algorithme réalisant ce calcul en respectant la complexité indiquée

- Reduce() : $\forall j \in \{1, \dots, p\}, N_j = \sum_{i=1}^{p} N_{ij}$. Déterminer k tel que $\sum_{j=1}^{k} N_j \leq \frac{\#E}{2} \leq \sum_{j=k+1}^{p} N_j$. $\mathbf{O(p^2)}$

Calcul de la médiane réelle :

- Map() : Sélectionner tous les éléments compris entre m_k et m_{k+1}. **O(n/p)**
- Reduce() : Déterminer la médiane réelle, qui se trouve parmi les éléments sélectionnés. $\mathbf{O}(\lceil \log_2(n/p) \rceil . n/p)$

Impact de l'organisation des données sur les temps d'exécution

Cas de la sélection

Soit une table T composée de n individus, ayant pour clef primaire un identifiant numérique. On cherche les données d'un individu donné.

Sans organisation particulière :

- Parcours séquentiel de la table jusqu'à trouver l'individu
- Temps en moyenne : O(n/2)
- Temps au pire : O(n)

Si T est triée par identifiant :

- On compare l'identifiant cherché avec l'identifiant médian. S'il n'est pas celui recherché, on répète l'opération récursivement sur le sous-tableau inférieur ou supérieur en fonction de l'ordre. (Algorithme dichotomique).
- Temps en moyenne : O(log2(n))
- Temps au pire : O(log2(n))

Impact de l'organisation des données sur la modélisation

Nous avons vu l'intérêt d'une organisation triée des données sur la performance des algorithmes informatiques.

Cependant, lorsque l'usage est la modélisation statistique, et sachant que la plupart des modèles usuels ne sont aujourd'hui pas adaptés à un développement parallélisé, il peut être souhaitable que le contenu de chaque nœud forme un échantillon représentatif de l'ensemble.

Exemple de méthode de modélisation compatible Map/Reduce :

Le bootstrap aggregating (ou bagging).

Map : p échantillons, répartis sur p nœuds, sont constitués à partir de l'ensemble d'apprentissage. La méthode de modélisation est appliquée sur chacun de ces échantillons.

Reduce :

- **Régressions :** Le modèle final est constitué par la moyenne des paramètres de chaque sous-modèle obtenu.
- **Discrimination :** Le modèle final est un voting des p sous-modèles obtenus.

Programme / Première partie

Le modèle client/serveur et ses limites

Des défis technologiques à résoudre pour passer la frontière du big data

- Préalable : quelques notions d'algorithmique
- Principe des architectures « big data »
- Organisation des données

Questions / réponses

Annexe : Code utilisé pour le test des tris (en langage C)

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <math.h>

long int tri_fusion(int * tab, const int deb, const int fin) {
   int mid,a,b,i;
   long int compteur;
   int * buf;
   if (fin-deb == 0 ) return 0; /* Un tableau à un élément est déjà trié */
   else {
      mid=deb+floor(fin-deb)/2;
      compteur=tri_fusion(tab,deb,mid); /* Tri récursif du demi-tableau de gauche */
      compteur+=tri_fusion(tab,mid+1,fin); /* Tri récursif du demi-tableau de droite */
      a=deb; b=mid+1;
      //Création d'un tableau pour les calculs intermédiaires
      buf=(int*)malloc(sizeof(int)*(fin-deb+1));
      for(i=0; i<=fin-deb; i++) { //Fusion des deux demi-tableaux, préservant le tri.
         if ( b>fin || (a<=mid && tab[a]<tab[b]) ) {buf[i]=tab[a]; a++;}
         else {buf[i]=tab[b]; b++;}
         compteur++;
      }
      //On copie le résultat final à sa place...
      memcpy(&tab[deb],buf,sizeof(int)*(fin-deb+1));
   }
   free(buf); //On libère la mémoire réservée pour les calculs intermédiaires
   return compteur;
}
```

```c
struct parm_tri_parallele {
int * tab, deb, fin; //Paramétres de l'algorithme
int max_threads; //Nombre max de fils d'exécutions parallèles à lancer (pour exploiter tous les coeurs de la machine)
int * nb_threads; //Compteur du nombre de fils lancés
long int res; //Compteur du nombre d'opérations réalisées par l'algorithme
};

void * tri_fusion_parallele(struct parm_tri_parallele * p) {
   int compteur, mid,a,b,i;
   int * buf;
   if (p->fin-p->deb == 0 ) {p->res=0; return NULL;} /* Un tableau à un élément est déjà trié */
   else {
      pthread_t t1, t2; int flag_t1=0, flag_t2=0;
      mid=p->deb+floor(p->fin-p->deb)/2;
      struct parm_tri_parallele p1=*p, p2=*p;
      p1.fin=mid;
      //Si on exploite déjà tous les coeurs du système, on fait un tri récursif classique
      if (*p->nb_threads>=p->max_threads) tri_fusion_parallele(&p1);
      //sinon, on lance le tri du sous-tableau dans un fil d'exécution parallèle
      else {(*p->nb_threads)++; flag_t1=1; pthread_create(&t1,NULL,tri_fusion_parallele,(void*)&p1);}
      p2.deb=mid+1;
      if (*p->nb_threads>=p->max_threads) tri_fusion_parallele(&p2);
      else {(*p->nb_threads)++; flag_t2=1; pthread_create(&t2,NULL,tri_fusion_parallele,(void*)&p2);}

      //Si des fils d'exécutions parallèles ont été lancés, on doit attendre qu'ils aient terminé leur travail pour démarrer la fusion
      if (flag_t1) pthread_join(t1,NULL);
      if (flag_t2) pthread_join(t2,NULL);
      compteur=p1.res+p2.res;

      //Fusion
      a=p->deb; b=mid+1;
      buf=(int*)malloc(sizeof(int)*(p->fin-p->deb+1));
      for(i=0; i<=p->fin-p->deb; i++) {
         if ( b>p->fin || (a<=mid && p->tab[a]<p->tab[b]) ) {buf[i]=p->tab[a]; a++;}
         else {buf[i]=p->tab[b]; b++;}
         compteur++;
      }
      memcpy(&p->tab[p->deb],buf,sizeof(int)*(p->fin-p->deb+1));
   }
   free(buf);
   p->res=compteur;
   return NULL;
}
```

Annexe : Code utilisé pour le test des tris (en langage C)

```
int tri_selection(int * tab, const int n) {
   int best, buf, i, j, compteur=0;
   for(i=0;i<n-1;i++) {
      /* best pointe le meilleur candidat trouvé */
      best=i;
      for (j=i+1;j<n;j++) { /* parcours des éléments non encore sélectionnés pour trouver le
meilleur candidat */
         if (tab[j]<tab[best]) best=j;
         compteur++;
      }
      /* On intervertit l'élément best avec l'élément i */
      buf=tab[best];
      tab[best]=tab[i];
      tab[i]=buf;
   }
   return compteur;
}
```

```
int test_fusion_parallele(const int n) {
 const clock_t temps_demarrage=clock();
  int * tab,i;
 tab=(int*)malloc(sizeof(int)*n);
 for (i=0;i<n;i++) tab[i]=rand();
 //for (i=0;i<n;i++) printf("%d\n",tab[i]);
 struct parm_tri_parallele p; int nb_threads=1;
 p.tab=tab; p.deb=0; p.fin=n-1; p.max_threads=4; p.nb_threads=&nb_threads;
 tri_fusion_parallele(&p); i=p.res;
 printf("*** FUSION PARALLELE %d threads *** Resultat en %d operations et %.3f
secondes\n",p.max_threads,i,((float)(clock()-temps_demarrage)/CLOCKS_PER_SEC));
 //for (i=0;i<n;i++) printf("%d\n",tab[i]);
 free(tab);
 return 0;
}
```

```
int test_fusion(const int n) {
 const clock_t temps_demarrage=clock();
  int * tab,i;
 tab=(int*)malloc(sizeof(int)*n);
 for (i=0;i<n;i++) tab[i]=rand();
 //for (i=0;i<n;i++) printf("%d\n",tab[i]);
 i=tri_fusion(tab,0,n-1);
 printf("*** FUSION *** Resultat en %d operations et %.3f secondes\n",i,((float)(clock()-
temps_demarrage)/CLOCKS_PER_SEC));
 //for (i=0;i<n;i++) printf("%d\n",tab[i]);
 free(tab);
 return 0;
}

int test_selection(const int n) {
 const clock_t temps_demarrage=clock();
  int * tab,i;
 tab=(int*)malloc(sizeof(int)*n);
 for (i=0;i<n;i++) tab[i]=rand();
 //for (i=0;i<n;i++) printf("%d\n",tab[i]);
 i=tri_selection(tab,n);
 printf("*** SELECTION *** Resultat en %d operations et %.3f secondes\n",i,((float)(clock()-
temps_demarrage)/CLOCKS_PER_SEC));
 //for (i=0;i<n;i++) printf("%d\n",tab[i]);
 free(tab);
 return 0;
}


void tester_tout(const int n) { printf("\n********** n = %d **********\n",n); if(n<500000)
test_selection(n); test_fusion(n); test_fusion_parallele(n);}

/* Fonction lancée à l'exécution du programme compilé */
int main() {
   tester_tout(1000);
   tester_tout(10000);
   tester_tout(100000);
   tester_tout(1000000);
   tester_tout(10000000);
   tester_tout(20000000);
   tester_tout(100000000);
   return 0;
}
```

Seconde partie

Le passage à l'échelle en pratique

BRIE PICARDIE

MDC
STRATÉGIE ET PILOTAGE
DE LA RELATION CLIENT

Programme / Seconde partie

- **Quand recourir à une architecture distribuée ?**
- **Etude de cas :** Modélisation numérique et intervalles de confiance sans hypothèse d'homoscédasticité
- Stockage et traitements batchs répartis / la solution Hadoop + Spark
- Mise en commun de compétences pluridisciplinaires par méthode agile
- **Questions / réponses**

BRIE PICARDIE

Les cas classiques de la mise en œuvre d'une archi distribuée…

- **La quantité de données à ingérer nécessite une répartition du stockage et des traitements** (ou anticipation d'une augmentation à ce niveau de besoin)

- **Réalisation d'économies d'échelles** par la mutualisation des ressources informatiques allouées à un grand nombre d'utilisateurs

En ai-je vraiment besoin ?

Avant d'envisager une architecture distribuée, il convient de se poser quelques questions…

- **Un échantillonnage ne suffit-il pas ?**
  - Ai-je vraiment besoin d'utiliser toutes les données ?
- **Mes programmes sont-ils optimisés ?**
  - Rapatriement de données non utiles (variables, historiques, libellés, …)
  - Complexité algorithmique
  - Jointures entre tables situées à des emplacements physiques ou logiques différents
  - Redondance de traitements (plusieurs fois la même requête, le même calcul intermédiaire, …)
- **Recherche du facteur limitant : qu'est-ce qui empêche mes traitement de se terminer plus rapidement ?**
  - Saturation du réseau entre le DWH et le serveur SAS ?
  - Débit disque / serveur SAS ?
  - Espace disque / serveur SAS ?
  - Puissance de calcul ?

Le mieux est l'ennemi du bien :

- Explorer les solutions les moins coûteuses
- On peut souvent se permettre de livrer une « V1 » non idéale, mais fonctionnelle

Programme / Seconde partie

- **Quand recourir à une architecture distribuée ?**
- **Etude de cas :** Modélisation numérique et intervalles de confiance sans hypothèse d'homoscédasticité
- Stockage et traitements batchs répartis / la solution Hadoop + Spark
- Mise en commun de compétences pluridisciplinaires par méthode agile
- **Questions / réponses**

Présentation du cas

Objectifs : Modéliser une donnée numérique (impôt sur le revenu), connue sur une partie du portefeuille client

Contraintes :

- La prédiction doit se faire sous forme d'intervalles de confiance,
- Les clients pour lesquels une prédiction précise est possible doivent être identifiés (pas d'hypothèse d'homoscédasticité)

Données d'apprentissage :

- 600.000 ménages ; 900 données par ménage

Moyens :

- 1 serveur SAS (qui a servi a construire la base d'apprentissage)
- Quelques machines linux équipés de Python et R

Une solution possible

- **Séparation de la base de données en plusieurs clusters de taille réduite**
- **Appliquer une méthode de régression cluster par cluster :** l'éventuelle hypothèse d'homoscédasticité n'a ainsi qu'une portée locale !
- **Evaluer les prédictions par validation croisée**
- **Passage à l'échelle (map) sur la partition formée par l'étape de clustering**
- **Réaliser une séparation discriminante plutôt qu'aléatoire :** Les sélections de variables, ainsi que la variance résiduelle, seront ainsi optimisées localement

Quelques bonnes pratiques

Optimiser l'organisation des données : en se posant les bonnes questions

Quel type de Map donnera le meilleur résultat ?

- Séquençage
- Echantillons aléatoires avec ou sans remise
- Classification métier
- Classification automatique

Viser la portabilité du code : que chaque unité aie un code identique !

Limiter les paramètre en dur dans le code (principaux pièges : configuration serveur, emplacements disque, …)

- Les paramètres communs à toutes les unités peuvent être stockés dans un fichier de configuration commun
- Les paramètres spécifiques à une unité (partitions, …) et/ou variables (dates, …) doivent être réduits le plus possible, et doivent pouvoir être générés dynamiquement par le batch principal.

Limiter le trafic réseau:

- Faire le moins possible de transports de données d'une machine à l'autre
- Déposer sur chaque machine toutes les données nécessaires à la réalisation de sa tâche (se méfier des disques partagés)

Programme / Seconde partie

- **Quand recourir à une architecture distribuée ?**
- **Etude de cas :** Modélisation numérique et intervalles de confiance sans hypothèse d'homoscédasticité
- Stockage et traitements batchs répartis / la solution Hadoop + Spark
- Mise en commun de compétences pluridisciplinaires par méthode agile
- **Questions / réponses**

hadoop : de quoi parle-t-on ?

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Site officiel du projet

http://hadoop.apache.org/

Un peu d'histoire :

Le développement de Hadoop fait suite à la publication par Google d'un article présentant sa solution de gestion de données massives : le Map-Reduce, avec un système de fichier adapté aux clusters Map-Reduce, le GoogleFS.

Le projet Hadoop a eu pour vocation première de proposer une implémentation open-source et en Java de ces technologies.

Architecture générale de hadoop

HDFS : Hadoop Distributed File System (inspiré de GoogleFS)

- Organise le stockage et les opérations de lecture, écriture, et gestion des données
- Répartit les données sur les différentes machines du cluster
- **S'assure à intervalle régulier que chaque machine fonctionne correctement et maintient un niveau paramétrable de réplication des données pour assurer la tolérance aux pannes**
- **Le tout de façon transparente pour l'utilisateur, qui accède à ses données sans se soucier de leur gestion**

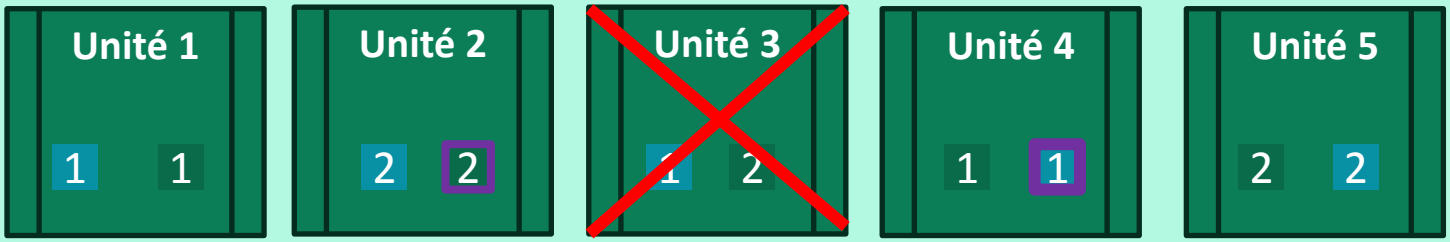
Serveurs de contrôle

NameNode primaire

NameNode secondaire

Optionnel
Prend le relai en cas de panne du primaire

Unités de calcul et de stockage (data nodes)



Un premier fichier à 2 blocs est créé

Un deuxième

L'unité 3 tombe en panne

Le NameNode le détecte et réagit

Les traitements / à bas niveau : Yarn

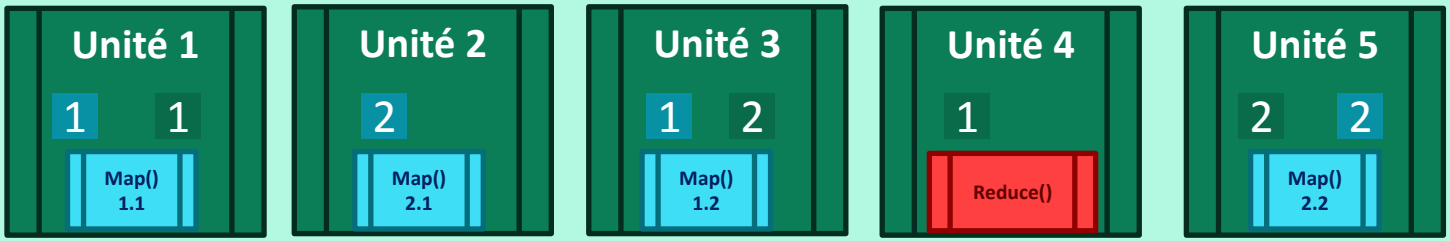
- Yarn est le module de Hadoop permettant de créer des jobs MapReduce
- Lorsqu'on crée un traitement Yarn, on définit un contexte (de quels fichiers on a besoin), et les commandes à lancer pour exécuter chaque Map() et Reduce()
- Yarn se charge de sélectionner les unités sur lesquels exécuter chaque traitement
- **Dans la programmation des jobs Yarn, on peut sélectionner certaines unités (pour réaliser les traitements là où sont les données !) ou prendre les premières unités disponibles (si on a juste besoin de puissance de calcul brute, où que l'on doit rassembler des données en provenance de plusieurs emplacements !)**

Serveurs de contrôle

Resource Manager

A chaque lancement de traitement, distribue la responsabilité d'exécuter les Map() et le Reduce().
Réalise le suivi, l'ordonnancement des différentes applications.
Le suivi de chaque application est délégué à l'unité chargé d'en faire le Reduce()

Unités de calcul et de stockage (node managers)



Un premier traitement est lancé sur tout le fichier bleu (par exemple une série de Map())

Un deuxième traitement est lancé sur une unité disponible (par exemple un Reduce())

En savoir plus :

- **Architecture Yarn : http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html**
- **Ecrire une application Yarn : http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/WritingYarnApplications.html**

Bibliothèques

Des modules pour Hadoop ont été développés pour un certain nombre de thèmes :

- **Mahout :** Datamining & Machine learning
- **HBase :** Gestion de données structurées sur HDFS
- **Hive :** Ajout de fonctions de type DWH (notamment, un langage de requêtes !)
- **Pig :** Langage permettant de distribuer des traitements de fichiers plats (logs, tables csv, …) sur un cluster hadoop, sans écrire de code Yarn / MapReduce !

Liste complète et références :

http://hadoop.apache.org/

Section « Related projects »

Les traitements / à haut niveau : Spark

- Des API Python, Scala, R, Java, pour le calcul réparti
- Un mappage à la volée et In-memory des données à traiter, via une structure de données distribuée, appelée RDD

```
def func_spark(num):
    path = "/tmp/" #Répertoir tmp des workers
    path_file = os.path.join(path, "test_ci"+str(num)+".txt")
    with open(path_file, "w") as testFile:
        _ = testFile.write(str(num)+" : "+str(gethostbyname(gethostname()))+"\n")
    os.system("hadoop fs -put "+path_file+ " "+"test_ci"+str(num)+".txt" ) # Copy de tmp à hdfs
    y=num #Numero de l'iteration
    t=gethostname() # Nom du hostname
    z=gethostbyname(gethostname()) # Adresse IP du hostname
    #print(os.listdir())
    return(y,t,z)
```

Exemple de code à distribuer

```
rdd=sc.parallelize([2,5,6,7,8])
```

Dataset partitionné à traiter

```
rdd.map(lambda x:func_spark(x)).collect()
```

Map / Collect !

```
17/10/31 18:30:12 INFO DAGScheduler: Job 2 finished: collect at <stdin>:1, took 7.143111 s
[(2, 'slzuyd5hsn04.yres.ytech', '10.166.60.147'), (5, 'slzuyd5hsn04.yres.ytech', '10.166.60.147'), (6, 'slzuyd5hsn
03.yres.ytech', '10.166.60.146'), (7, 'slzuyd5hsn03.yres.ytech', '10.166.60.146'), (8, 'slzuyd5hsn03.yres.ytech',
'10.166.60.146')]
>>>
```

L'API Spark

Quelques fonctions courantes de l'API Python (pyspark) opérant sur RDD :

Transformations :

RDD->RDD

- **map()**
- **flatMap()**
- **filter()**
- **sample()**
- **union()**
- **intersection()**
- **distinct()**
- **join()**
- **reduceByKey()**

Actions:

RDD->Résultat

- **reduce()**
- **collect()**
- **count()**
- takeSample(), countByKey(), …

Partitionnement :

- **coalesce()**
- **repartition()**
- **persist()**

Référence : https://spark.apache.org/docs/latest/api/python/pyspark.html

Bibliothèques

Des modules pour Spark :

- **MLLib :** Datamining & Machine learning
- **Mais aussi tout module python :** Numpy + sklearn, tensorflow, …

Liste complète et références :

http://spark.apache.org/
Section « Libraries »

Limites

Formation !

Les ingénieurs formés à ces technologies restent aujourd'hui difficiles à trouver sur le marché

Gestion des données

- Le SQL ne suffit plus! (*individus / variables*)
- Il faut apprendre à raisonner ***partitions*** */ individus / variables*
- Les API Spark distribuent **implicitement** : c'est productif et efficace en nombre de lignes de code, mais complexe dans la compréhension des mécanismes implicites et de leurs impacts sur les flux / stockages / localisations des données.

Caractère pluri disciplinaire !

Pour utiliser Hadoop/Spark dans un cadre datamining / stats, il est nécessaire de parvenir à réunir et coordonner des compétences de pointe :

- **en mathématiques :** repenser les modèles en mode distribué (choisir les bonnes clefs de partitionnement)
- **en informatique :** maîtrise du modèle en cluster (impacts sécurité, déploiement, maintenance), de la programmation map/reduce, …

Programme / Seconde partie

- **Quand recourir à une architecture distribuée ?**
- **Etude de cas :** Modélisation numérique et intervalles de confiance sans hypothèse d'homoscédasticité
- Stockage et traitements batchs répartis / la solution Hadoop + Spark
- Mise en commun de compétences pluridisciplinaires par méthode agile
- **Questions / réponses**

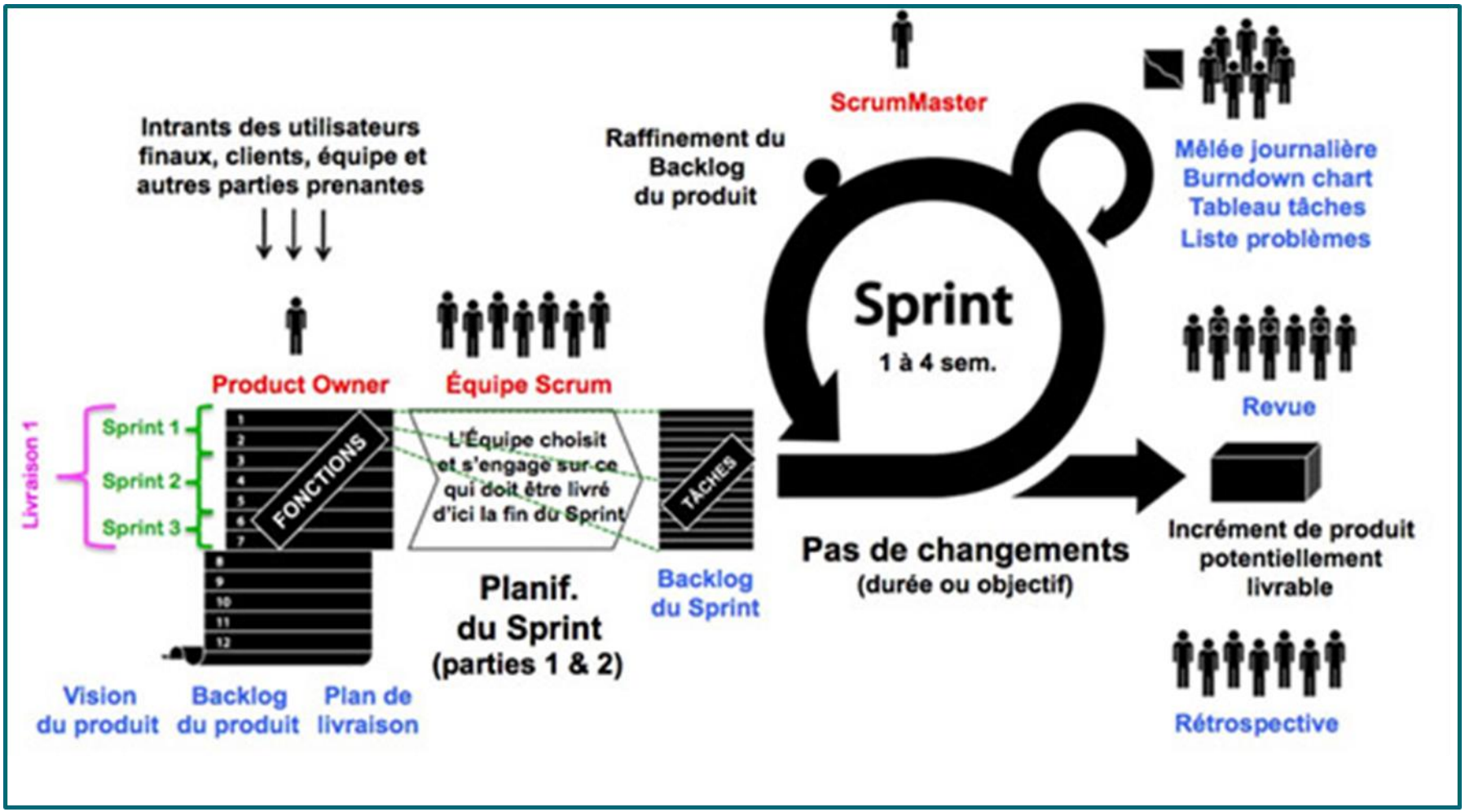
Les difficultés organisationnelles du big data

Les projets Big Data posent souvent les problèmes suivants :

- Nécessité de constituer une équipe de développement pluridisciplinaire (statistiques, calcul distribué, métier, …)
- La spécialisation de chacun complexifie la communication (risque d'isolement)
- Effet tunnel => Risque élevé de ne détecter les défauts de conception qu'en fin de projet
- Maintenabilité, qualité et propriété collective du code réduites par le manque de polyvalence

La gestion de projet classique, « en V », voit ses défauts exacerbés sur les projets data !

Exemple de méthode projet adaptée : Scrum



Découpage du produit en besoins : le backlog du produit

Exemple sur un projet de segmentation

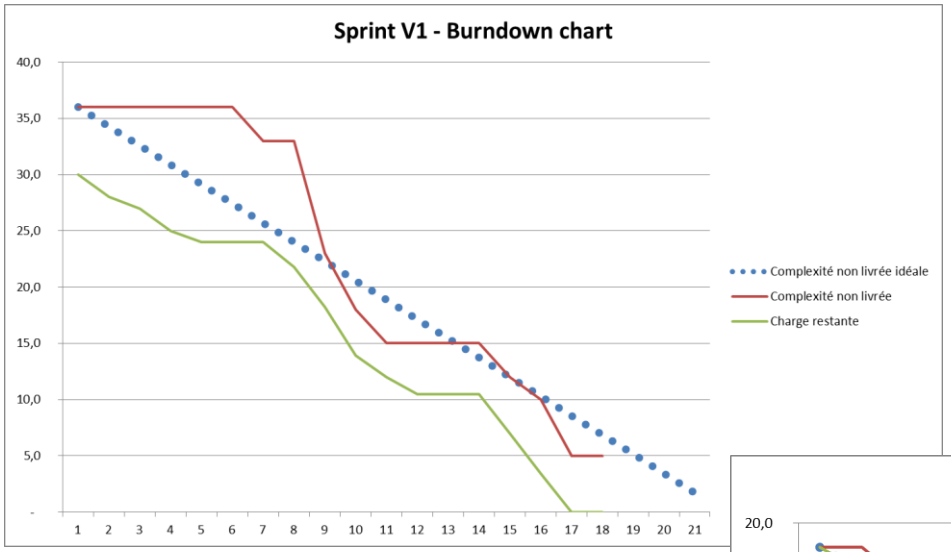
Lot	Description	Détail	VOR	Ordre	Itération prévis.	Objectif	Complexité initiale	Charge effective en j	Statut
Prérequis	Organisation fonctionnement équipe	Règles (mode projet), outils, normer la doc	V	1	0.1	Sprint 0	2	1,3	Livré
Prérequis	Structure technique (couches)	Structurer l'architecture algorithmiques en couches (au minimum : 1=>Extraction données 2=>Création briques de connaissance client 3=>Assemblage 4=>Restitution et statistiques)	V	2	0.1	Sprint 0	1	0,5	Livré
Prérequis	Création du périmètre	Ecriture des couches Extraction/Brique du périmètre (liste des id. des relations clients à segmenter ou au contraire à exclure) + Cleaning anciens clients et prospects	V	3	0.1	Sprint 0	8	5,5	Livré
Prérequis	Création du paramétrage de restitution des segments	Ecriture de la couche d'assemblage / spécification des fichiers de paramétrage	V	4	0.1	Sprint 0	21	14,3	Livré
Périmètre	Segment nouveaux clients		V	10	0.4	1	1	0,3	Livré
Seg CASA	Vision unifiée	Calcul des indicateurs consolidées à la relation (intégration des SCI locatives)	V	15	0.4	1	8	6,3	Livré
Seg CASA	Stabilisation segments	Création notions de flux et stocks stables (stabilité à 3 mois)	V	20	0.4	1	2	2	Livré
Seg CASA	Stabilisation segments	Application à la stabilisation des segments HDG et CI	V	21	0.4	1	1	1,4	Livré
Potentiel extérieur	RFR	Intégration du RFR dans les règles d'assemblage	V	90	0.9	1	2	6,4	Livré
Technique	Correctifs	Passage à l'id_relation de toutes les données encore au niveau CC / intégration du contrôle relatif à la segmentation CASA issue du système d'informations	O	90	0.9	1	0	7,6	Livré
Potentiel extérieur	Multi-bancarisation	Exploitation de l'indicateur fidélité CSP + tests	V	91	0.9	1	5	2,5	Livré
Comportement multicanal	Nb connexions BAM		V	100	1.0	1'	2	2,1	En cours
Comportement multicanal	Nb connexions ma banque		V	100	1.0	1'	2	2,1	En cours
Comportement multicanal	Opérations agences		O	100	1.0	1'	5	3,6	En cours
Comportement multicanal	Nb RAC Agence		O	105	1.0	1'	2	1,5	En cours
Comportement multicanal	Contrôle après assemblage		O	109	1.0	1'	13	0,8	En cours
Restitution	Rapports automatiques de restitution (première proposition rustique)	bonne exécution / anomalies / compteurs / éventuellement un TDB de synthèse automatique		109	1.0	1'	0	4,1	En cours
Restit Maille Maillage	Structure des restitutions à l'agence gestionnaire			110	1.8	M1	1		Non démarré
Restit Maille Maillage	Centralisation / Importation données externes	Centralisation manuelle faite par Jeremy pour V1		112	1.8	M1	3		Non démarré

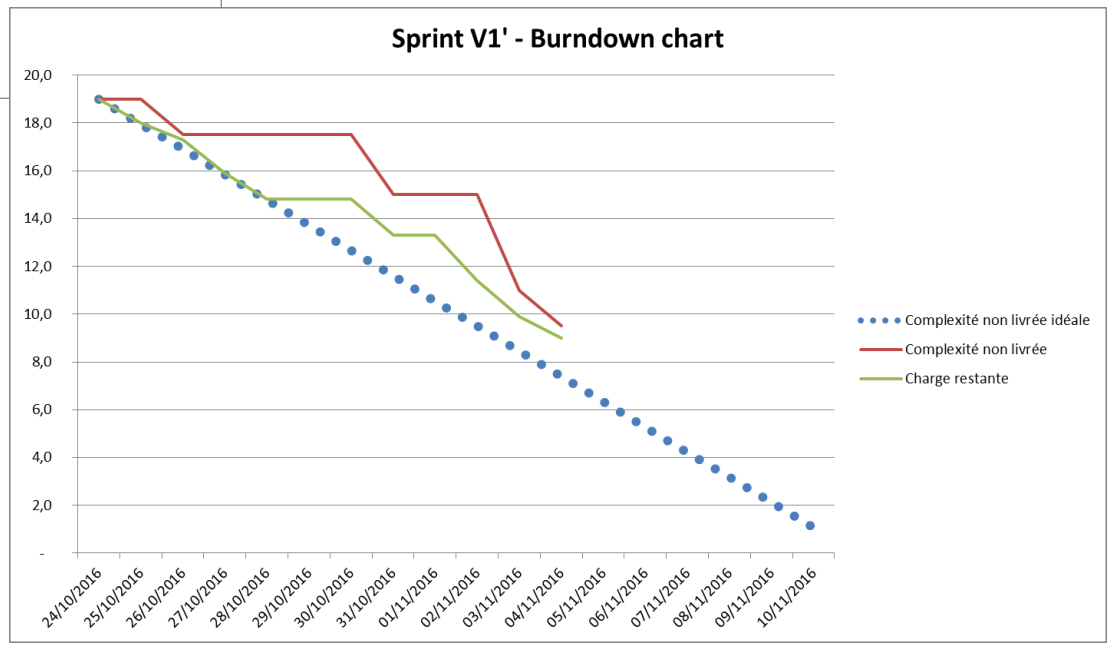
Suivi de l'avancement du sprint: sprint backlog

Description	Détail	Tâches	VOR	Date début prévue	Date fin prévue	Retard ?	Statut	Complexité (pts)	Charge effective en j	Personne(s) en lead	Commentaire
GLOBAL SPRINT V1			V	**VELOCITE (pts/j):**			1,25	**36,0**	**26,5**		
GLOBAL ITERATION 0.4			V					**13,0**	**10,0**		
Segment nouveaux clients	Segment nouveaux clients		V	17/10/2016	17/10/2016	-	Livré	1,0	0,3	Pierre	
						-					
Vision unifiée	Calcul des indicateurs consolidées à la relation (intégration des SCI locatives)	Expérimentation / découverte des données	V	29/09/2016	03/10/2016	-	Livré	2,0	2,2	Estelle	
		Décisions / spécifications détaillées du livrable	V	30/09/2016	05/10/2016	-	Livré	2,0	2,0		
		Développement informatique	V	06/10/2016	07/10/2016	-	Livré	2,0	1,7		
		Intégration / création du plan de test automatique	V	20/10/2016	25/10/2016	-	Livré	2,0	0,4		
Stabilisation segments HdG / CI	Création notions de flux et stocks stables (stabilité à 3 mois)		V	17/10/2016	19/10/2016	-	Livré	2,0	2,0	Lydie	
						-					
						-					
	Application à la stabilisation des segments HDG et CI		V	17/10/2016	19/10/2016	-	Livré	2,0	1,4	Lydie	
						-					
						-					
GLOBAL ITERATION 0.9			V					**23,0**	**16,5**		
RFR	Intégration du RFR dans les règles d'assemblage	Découverte	V	19/10/2016	28/10/2016	-	Livré	2,0	2,1	Igor	
		Spécifications	V	17/10/2016	19/10/2016	-	Livré	1,0	0,5		
		Mise en œuvre one shot	V	18/10/2016	21/10/2016	-	Livré	2,0	2,0		
		Documentation pour industrialisation future	V	20/10/2016	25/10/2016	-	Livré	2,0	1,8	Igor	
Multibancarisation	Exploitation de l'indicateur fidélité CSP + tests	Découverte	V	16/10/2016	19/10/2016	-	Livré	1,0	0,9	Estelle	
		Spécifications	V	18/10/2016	20/10/2016	-	Livré	1,0	0,5		
		Mise en œuvre	V	21/10/2016	25/10/2016	-	Livré	1,0	0,4		
		Industrialisation / contrôle	V	21/10/2016	25/10/2016	-	Livré	2,0	0,7		
Technique	Remboursement dette technique accumulée en 0.1 et 0.4	Passage à l'id_relation de l'âge	V	14/10/2016	18/10/2016	-	Livré	1,0	0,9	Lydie	
		Passage à l'id_relation des données de contrôle ADDCCO	V	14/10/2016	18/10/2016	-	Livré	1,0	0,5		
		Assemblage à l'id_relation	V	14/10/2016	20/10/2016	-	Livré	3,0	5,2	Lydie	
		Parametrage : Devpt macro concordance Excel / CSV + dossier caché pour les CSV + 1iere ligne NE PAS MODIFIER ICI dans CSV + import à partir de la 3ieme ligne dans SAS	V	24/10/2016	28/10/2016	-	Livré	1,0	0,2	Estelle	
		Assemblage/Dérivation : prioriser l'exécution sur la priorité de chaque segmentation plutôt que prioriser en dur dans le code la segmentation CASA	V	24/10/2016	28/10/2016	-	Livré	2,0	0,5	Igor	DETTE TECHNIQUE acceptée pour l'instant / résorbtion à planifier sur un sprint ultérieur ; ou à reprendre en fin de sprint s'il reste du temps
		Contrôles / mise en place des tests unitaires	R	20/10/2016	28/10/2016	-	Suspendu	3,0	0,3		

Suivi de l'avancement du sprint: burndown chart

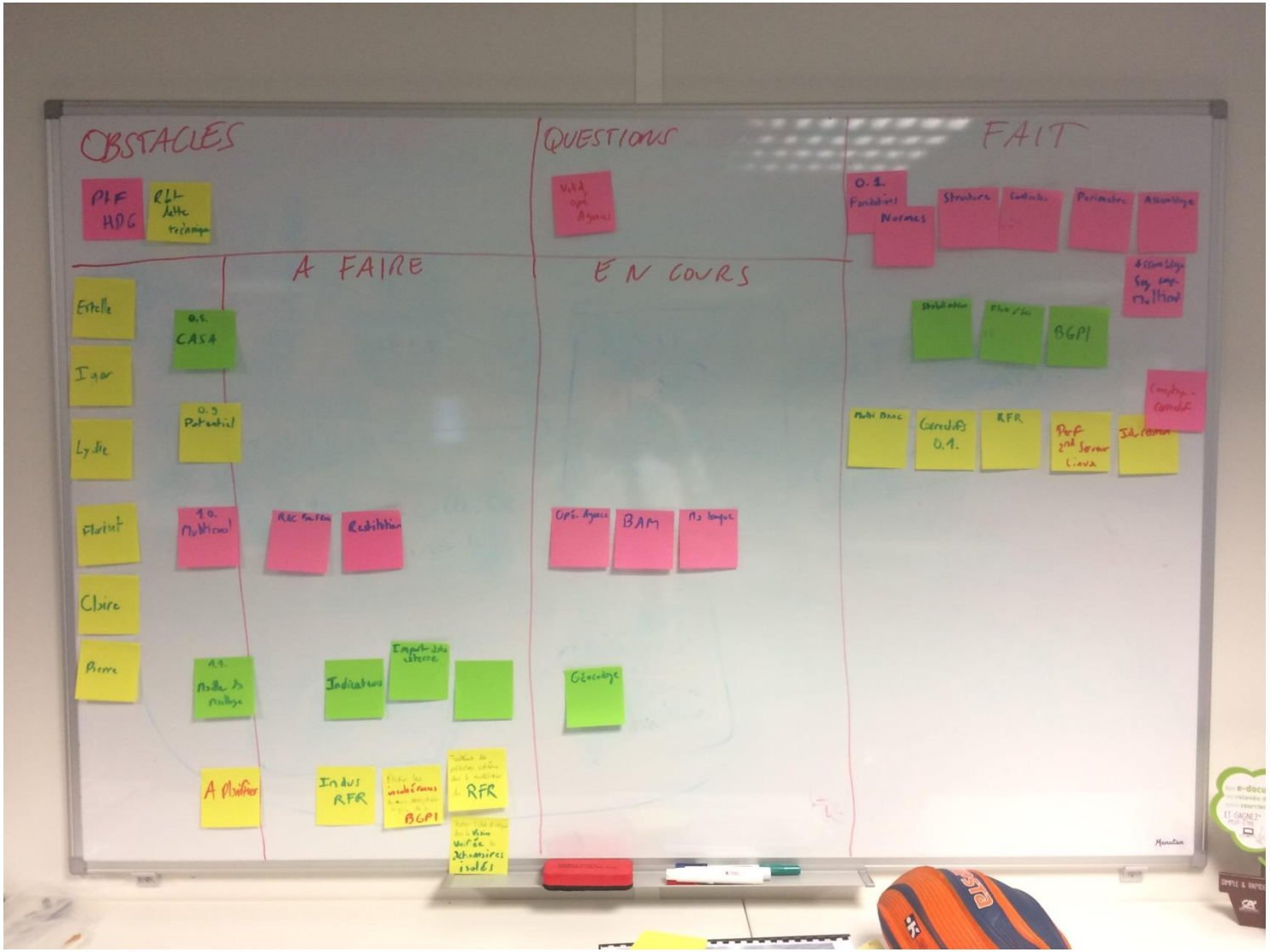
Courbes d'un sprint « OK »





Courbes d'un sprint en léger décalage

Pour la mêlée quotidienne : le tableau des tâches



Programme / Seconde partie

- **Quand recourir à une architecture distribuée ?**
- **Etude de cas :** Modélisation numérique et intervalles de confiance sans hypothèse d'homoscédasticité
- Stockage et traitements batchs répartis / la solution Hadoop + Spark
- Mise en commun de compétences pluridisciplinaires par méthode agile
- **Questions / réponses**