C-Wire

Par :

Jean-Luc MASLANKA
Ibrahima BALDÉ-CISSÉ
Gaspard SAVÈS

Classe de :

Pré-ingénieur, deuxième année

Mathématiques-Informatique Économie-Finance 2, Groupe B

Sous la direction de :

Romuald GRIGNON, enseignant-chercheur en informatique

Le 22 décembre 2024

Année 2024 - 2025

SOMMAIRE

SOMMAIRE..... 2
Introduction..... 3
I – Répartition des tâches et calendrier de réalisation..... 3
A - Organisation générale..... 3
B - Planning..... 4
II – Phase de Développement..... 4
A - Compréhension et Préparation..... 4
B - Premières implémentations..... 4
C - Tests et optimisation..... 5
D - Finalisation, bonus et rédaction des documents utilisateurs..... 6
III - Explication des fichiers de sortie et du répertoire test..... 7
A - Fichiers de sorties..... 7
B - Fichiers tampons..... 9
C - Histogrammes LV..... 10
D - Fichier PDF des histogrammes LV..... 12
Conclusion..... 12

Ce rapport vise à présenter l'évolution de notre travail sur ce projet, la répartition des tâches, les problèmes que nous avons rencontrés et les solutions que nous y avons apportées. Les différentes phases y sont résumées de manière succincte et des exemples précis de fichiers de sorties sont explicités à la fin du présent rapport.

I – Répartition des tâches et calendrier de réalisation

A - Organisation générale

Nous avons commencé à travailler pleinement sur le projet à partir du premier décembre car nous avions beaucoup d'examens avant. Nous organisions notre travail autour des trois séances de TD de la semaine et réalisions une visioconférence chaque dimanche après-midi pour orienter les prochains travaux et faire le point sur ce qui avait pu être implémenté et les problèmes que nous devions résoudre. Cette organisation ainsi que notre usage d'un dépôt GitHub couplé à VS Code nous permettait de limiter les problèmes liés à la synchronisation des différentes tâches.

Après plusieurs discussions nous avons décidé de se répartir le travail de la manière suivante :

- Gaspard s'est occupé de la partie Shell du projet,
- Ibrahima et Jean-Luc se sont occupés du code C

Pour les bonus :

- Ibrahima s'est chargé de la partie GnuPlot
- Jean-Luc s'est chargé de la partie LaTeX
- Et Gaspard s'est occupé de l'implémentation générale dans le projet

Malgré cette répartition des tâches nous avons travaillé régulièrement ensemble via Live Share (outil qui permet de coder en voyant en temps réel les modifications des autres collaborateurs) ou en présentiel pour résoudre les problèmes ou implémenter le travail des autres.

B - Planning

Le carnet de bord du dépôt permet de constater la progression que nous avons eue et les principales modifications qui ont été implémentées jour après jour. (S'y référer pour plus de détail)

II – Phase de Développement

A - Compréhension et Préparation

Pour cette phase initiale, l'équipe s'est réunie peu après avoir eu le sujet afin d'établir notre stratégie de réalisation, les différents fichiers de code que nous ferons et nos structures. Malgré une première impression de complexité, nos questions au professeur nous ont permis de nous rendre compte que le projet était moins difficile qu'envisagé et nous comprenons qu'une seule structure de données sera nécessaire pour l'AVL en rédigeant un code C généraliste.

B - Premières implémentations

La deuxième phase a marqué le début de l'implémentation technique. Le code C avec les arbres types AVL et leurs mécanismes d'équilibrage et de rotation, ont été développés par Ibrahima et Jean-Luc, tandis que le script *Shell* est écrit par Gaspard.

L'objectif principal pour Gaspard était d'arriver le plus rapidement possible à développer le tri des données et une analyse robuste des arguments pour pouvoir tester le bon fonctionnement du code C en conditions réelles.

Cependant, des défis ont rapidement émergé, notamment un temps de traitement trop élevé, Gaspard comprend rapidement que les boucles *Shell* sont inutilisables (temps de traitement > 30 minutes pour 'hva comp' sur la v25) et que les fichiers tampons volumineux sont à bannir (temps de traitement de 27 secondes pour 'hva comp' sur la v25). Par conséquent, il est décidé d'optimiser le script en utilisant la commande *grep* et de "piper" directement les données dans le C pour éviter les fichiers tampons. Ces optimisations permettent d'arriver à un traitement d'1 seconde seulement pour 'hva comp' et de 15 secondes pour 'lv all' avec le traitement LV Min/max.

Cette étape voit aussi l'implémentation du tri par numéro de centrale et le nommage des fichiers de sortie pour répondre aux attendus. Le tri par centrale est effectué grâce à une variable initialisée à une valeur neutre [0-9]+ (qui accepte toutes les centrales) lorsque cette option n'est pas activée, ce qui évite de dupliquer du code.

C - Tests et optimisation

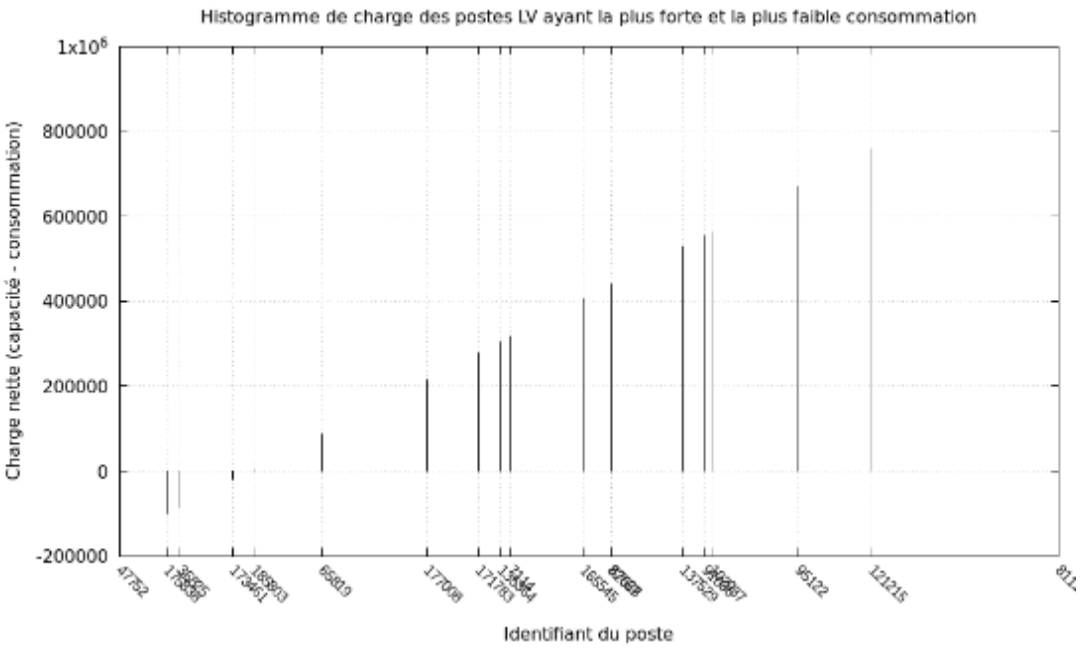
La troisième phase a été consacrée à des tests plus poussés et l'optimisation des performances.

Après avoir observé ce temps de traitement de 15 secondes pour la commande 'lv all' alors même qu'il n'y avait pas encore la construction des histogrammes, nous décidons de faire la "chasse aux secondes" pour améliorer encore nos performances. Nous réussissons à gagner 1 seconde en simplifiant l'expression régulière de la commande grep et 3 secondes grâce à la brillante idée de Jean-Luc de revoir et d'optimiser la contrainte d'alignement et le padding de la structure de notre AVL. Cette optimisation a sûrement permis des allocations mémoire plus rapides lors du traitement par le code C.

C'est également lors de cette phase que nous avons intégré l'histogramme Min/max dans le cadre de la commande 'lv all' grâce au script GnuPlot rédigé par Ibrahima.

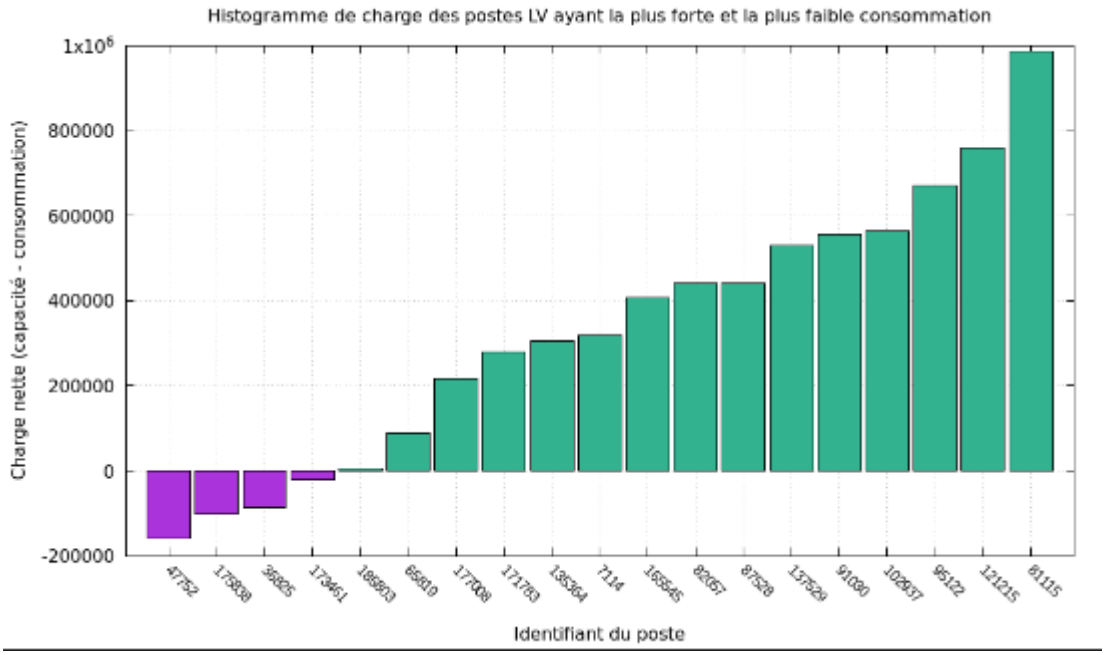
Quelques problèmes vont alors nous donner du fil à retordre… Tout d'abord, l'échelle prise en compte sur l'axe des abscisses qui ne doit pas être celle des identifiants des postes LV car sinon le graphique est illisible.

Exemple de graphique avec la mauvaise échelle :



Nous arrivons à régler ce problème d'échelle et réalisons un premier graphique acceptable.

Premier graphique généré par le script GnuPlot :



Toutefois, nous considérons que ce premier graphique n'est pas forcément très clair car il ne fait ressortir que la consommation excédentaire ou la consommation supplémentaire. De plus, lorsqu'il est fait avec les postes LV les plus chargés et les moins chargés, il est tout bonnement illisible à cause de l'échelle. Par conséquent, nous décidons de séparer le script en deux pour réaliser deux graphiques, l'un avec les postes en sous-charge et l'autre avec les postes en surcharge. Le résultat final est beaucoup plus clair (cf. partie III C du présent rapport).

Nous avons, à ce moment-là, un temps de traitement pour 'lv all' qui est d'environ 10 secondes sur nos machines, l'implémentation des graphiques Gnuplot a rallongé légèrement ce temps de traitement d'environ 1 seconde.

D - Finalisation, bonus et rédaction des documents utilisateurs

La dernière semaine a consisté en une amélioration de notre application fonctionnelle. L'ensemble des attendus fonctionnels étaient déjà au rendez-vous, nous procédons donc à des améliorations tel que la réduction et la clarification des affichages dans le terminal lors de l'exécution, la réalisation d'un document LaTeX contenant l'ensemble des histogrammes LV ainsi qu'à la réalisation du README en Markdown et ce rapport. Ce temps a permis la livraison d'une version beaucoup plus propre et aboutie du projet et à la production de documents explicatifs qui facilitent la vie de l'utilisateur.

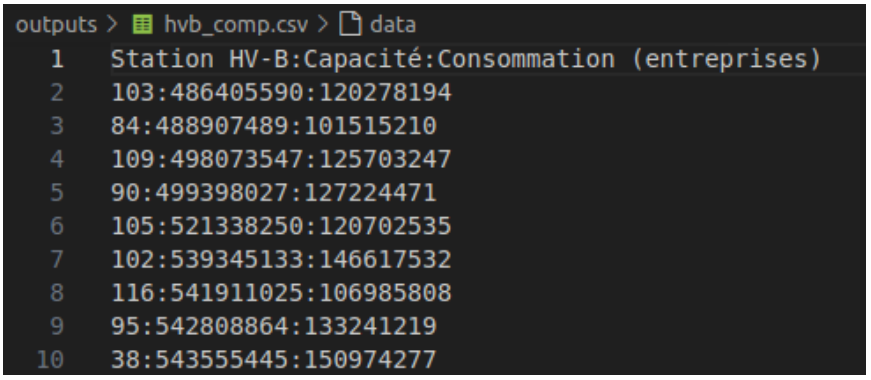
III - Explication des fichiers de sortie et du répertoire test

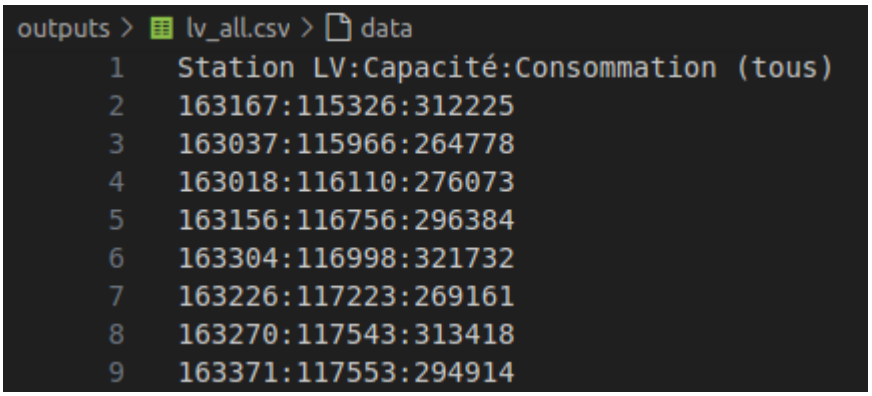
Le répertoire ‘tests’ contient de nombreux fichiers de sortie qu’il est possible de reproduire en exécutant le script shell. Vous y trouverez différents types de documents qui sont explicités dans les sous parties suivantes. Ils sont répartis dans deux sous-répertoires ‘v00’ et ‘v25’ en fonction du fichier de base qui a été utilisé pour les générer.

A - Fichiers de sorties

L'exécution du script renvoie au moins un fichier .csv de trois colonnes en sortie avec l’identifiant du poste, sa capacité, et la somme des consommations des consommateurs approvisionnés par ce poste. Les postes (que ce soit HV-B, HV-A ou LV) sont alors classés par capacité croissante.

Exemples de l’entête de deux fichiers de sortie avec les commandes ‘HVB comp’ et ‘LV all’

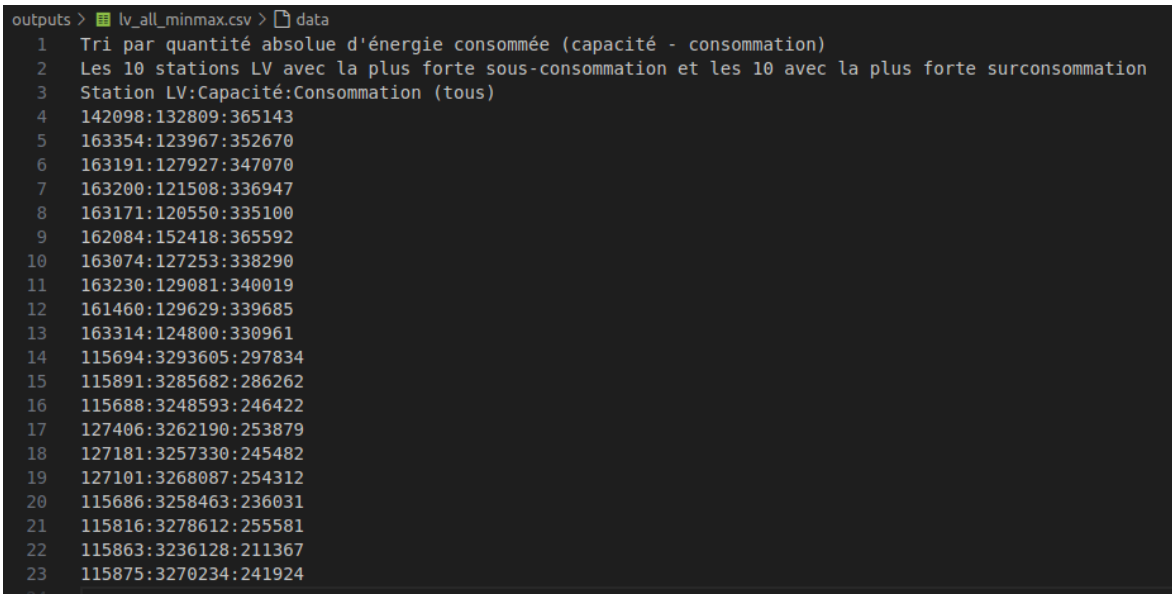




Dans le cadre d’un traitement LV all, un fichier supplémentaire LV Min/max est généré si le fichier de sortie initial LV all contient plus de 20 postes. Ce fichier contient les 10 postes les

plus chargés et les 10 postes les moins chargés c'est-à-dire les postes pour lesquels la valeurs de la différence entre capacité et consommation est la plus basse ou la plus haute. Les postes sont alors classés du plus chargé (capacité – consommation le plus petit) au moins chargé (capacité – consommation le plus grand).

Exemple de fichier LV Min/max obtenu avec la v25 :



B - Fichiers tampons

Nous avons fait le choix de minimiser au maximum l'usage des fichiers tampons, il y en a donc uniquement dans le cadre du traitement LV Min/max car ils permettent de calculer et de classer les postes par niveau de charge (capacité – consommation) et de les trier en fonction de ce ratio. Ils nous permettent ensuite de scinder en deux les données entre postes en sous-charge et poste en surcharge avant de les envoyer dans leurs scripts GnuPlot respectifs.

Premier fichier tampon qui est généré après le calcul de la charge

tmp > buff_lv_all_minmax.csv > data

```
142098:132809:365143:-232334
163354:123967:352670:-228703
163191:127927:347070:-219143
163200:121508:336947:-215439
163171:120550:335100:-214550
162084:152418:365592:-213174
163074:127253:338290:-211037
163230:129081:340019:-210938
161460:129629:339685:-210056
163314:124800:330961:-206161
115694:3293605:297834:2995771
115891:3285682:286262:2999420
115688:3248593:246422:3002171
127406:3262190:253879:3008311
127181:3257330:245482:3011848
127101:3268087:254312:3013775
115686:3258463:236031:3022432
115816:3278612:255581:3023031
115863:3236128:211367:3024761
115875:3270234:241924:3028310
```

Deuxième et troisième fichier tampons contenant les postes en sous-charge ou ceux en surcharge qui seront envoyés aux deux scripts Gnuplot :

tmp > buff_plt_lv_all_minmax_underload.csv

```
115863:3236128:211367
115686:3258463:236031
115875:3270234:241924
127181:3257330:245482
115688:3248593:246422
127406:3262190:253879
127101:3268087:254312
115816:3278612:255581
115891:3285682:286262
115694:3293605:297834
```

tmp > buff_plt_lv_all_minmax_overload.csv

```
163171:120550:335100
163200:121508:336947
163354:123967:352670
163314:124800:330961
163074:127253:338290
163191:127927:347070
163230:129081:340019
161460:129629:339685
142098:132809:365143
162084:152418:365592
```

C - Histogrammes LV

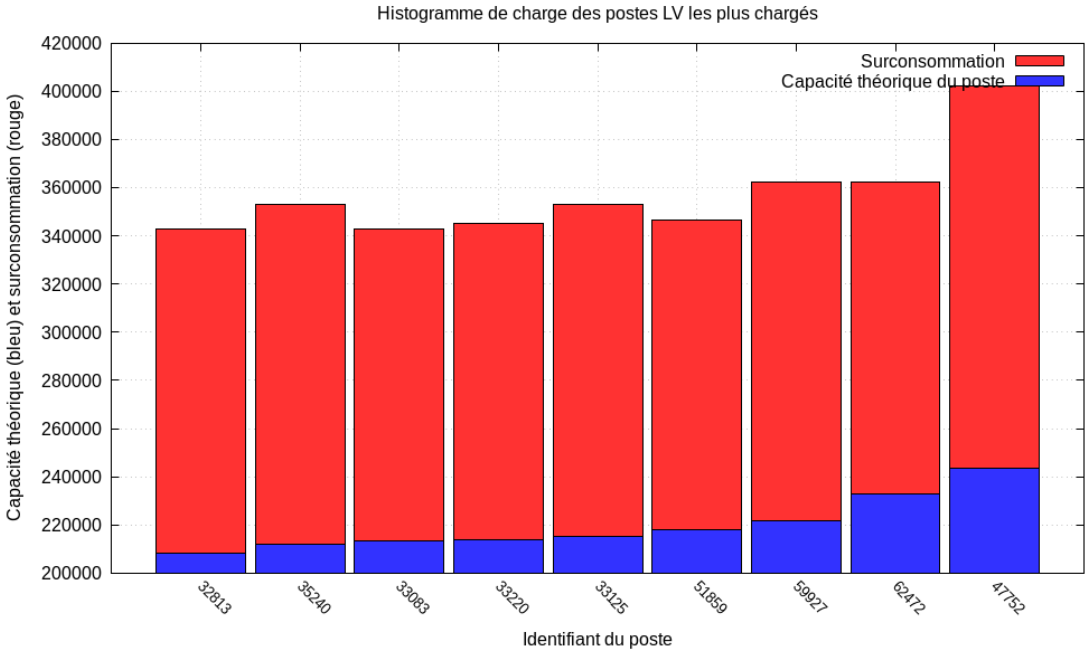
Pour l'association de commandes 'lv all', 2 graphiques sont donc générés, le premier traite les postes LV en surcharge (capacité – consommation < 0) et le second traite les postes LV en sous-charge (capacité – consommation ≥ 0). (Par convention, les postes dont la capacité est égale à la consommation seront considérés comme en sous-charge)

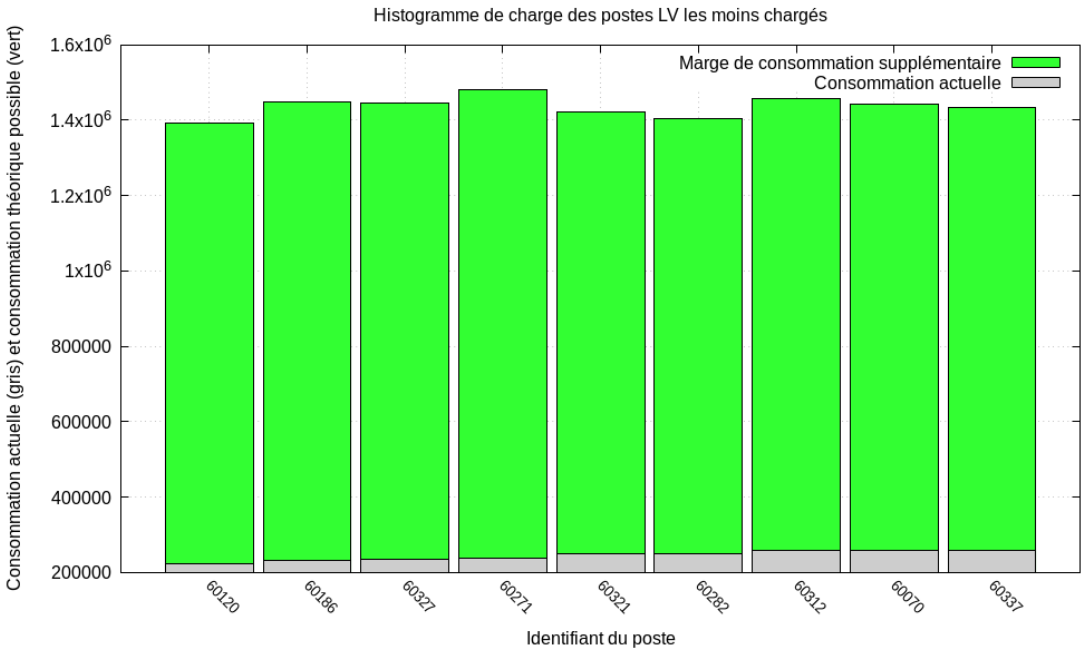
Dans l'histogramme des postes en surcharge (celui dont le nom finit par 'overload'), la zone en rouge représente la consommation excédentaire des postes en surcharge et en la zone en bleu la capacité théorique de chaque poste.

Dans l'histogramme des postes en sous-charge (celui dont le nom finit par 'underload'), la zone grise représente la consommation actuelle et la zone verte la consommation supplémentaire possible au vu de la capacité du poste.

Le nombre de barres s'adapte automatiquement en fonction du nombre de postes à traiter dans les deux scripts Gnuplot.

Voici un exemple avec les LV approvisionnés par la centrale 2 du fichier v25.





D - Fichier PDF des histogrammes LV

Le fichier pdf généré dans le dossier ‘output’ a pour but de rassembler les histogrammes dans un même document pour faciliter leur analyse.

Notre projet final est composé à 60% de *Shell* et à 26 % de *C*, le reste étant réparti entre *GnuPlot*, *LaTeX* et *makefile* ; aucun bug n’est à signaler. Les fichiers retournés correspondent aux attentes et le programme s’exécute en 12 secondes environ pour la commande ‘lv all’ et 14 secondes pour ‘lv indiv’ qui est la plus longue en temps de traitement.