

Recherche & Développement autour de la problématique du Big Data



VIGNERON

Joris



□ Sommaire

1. Introduction
2. Présentation de l'entreprise
3. Sujet du stage
4. Etat de l'art
5. Découverte et expérimentation
6. Application
7. Conclusion

Introduction

- Choix de l'Observatoire astronomique
- 2 raisons ayant motivé ce choix

Présentation de l'entreprise

- Stage réalisé à l'Observatoire astronomique de Strasbourg
- Construit en 1881, possède la 3^{ème} plus grande lunette de France par sa taille
- Il se doit de contribuer aux progrès de la connaissance



Observatoire astronomique
de Strasbourg

Présentation de l'entreprise – Les équipes de recherche

- Galaxies : réalise des recherches sur la formation des galaxies
- Hautes énergies : s'intéresse aux sources émettrice de rayon X
- Le CDS : développe des services d'accès aux données



Présentation de l'entreprise - Les services du CDS



Base de données de référence pour la bibliographie des objets astronomiques

Plus de 40 ans de données, informations sur plus de 8 Millions d'objets

Base de données qui regroupe plus de 13 000 catalogues d'objets célestes. 300 000 requêtes par jour en moyenne.

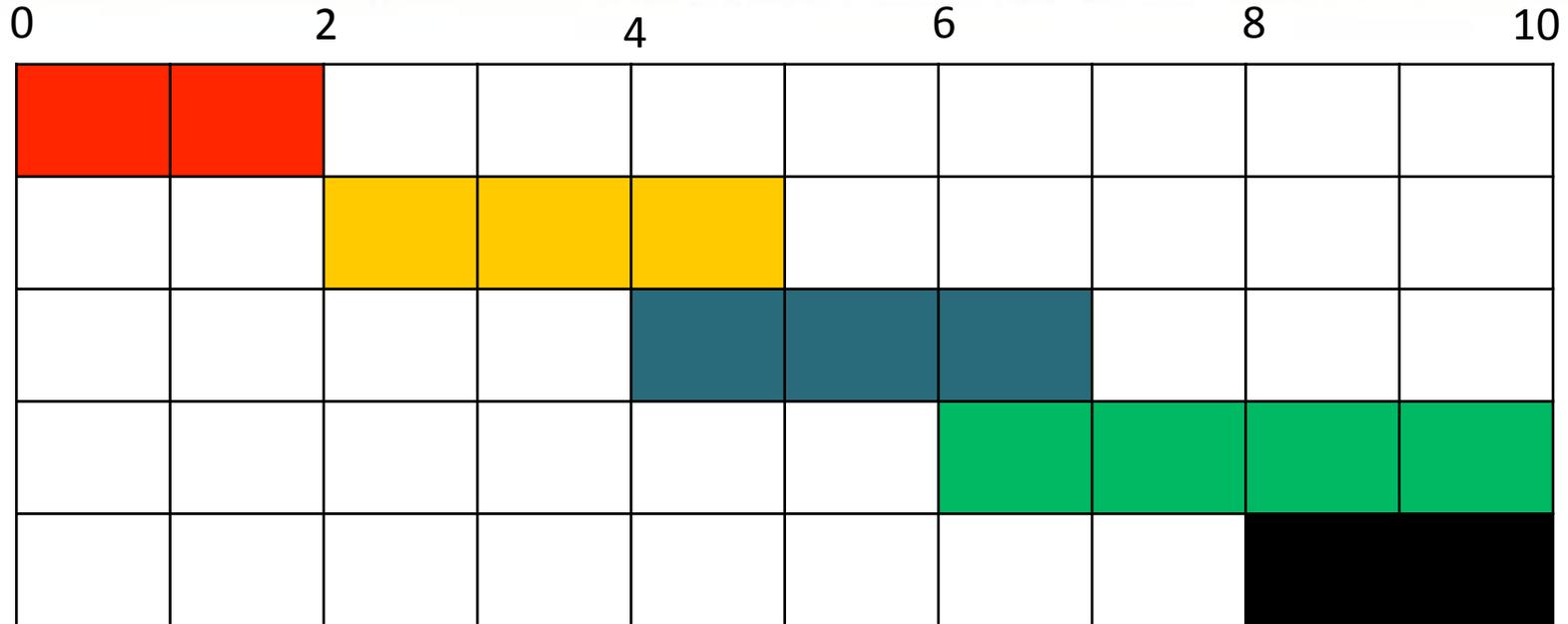


Atlas interactif du ciel permettant de visualiser des images astronomiques

Sujet du stage

- Problématique du Big Data et notamment les technologies qui gravitent autour
- Enjeux du stage :
 - Réalisation de diverses expérimentations dans le but d'éprouver les outils à disposition
 - Mise en œuvre d'un cas d'utilisation de l'Observatoire

Sujet du stage - Déroulement



Etat de l'art



Essais avec Spark



Rédaction du rapport



Essais avec Hadoop



Réalisation de l'application

Etat de l'art

- Début du stage : découvrir les technologies du Big Data
- Apprentissage des différentes facettes de cette technologie

Etat de l'art – Le Big Data

- Big Data ou « grosses données »
- L'humanité génère 2,5 trillions d'octets de données/jours
- Les 3V : Volume, Variété, Vitesse
- Faiblesse des BD classiques



Etat de l'art - MapReduce

- Patron d'architecture de programmation
- Manipulation de grandes quantités de données
- Fonctionnement en 2 phases : phase Map et phase Reduce

Etat de l'art - Hadoop

- Framework Java Open Source développé par la fondation Apache



- Hadoop Distributed File System, HDFS est le système de fichier distribué d'Hadoop
- Basé sur le patron MapReduce
- Différentes distributions implémentant ce framework

Etat de l'art - Spark

- Framework développé par AMPLab de l'université de Berkeley
- Modèle de programmation plus simple et temps d'exécution 100 fois plus courts
- Le RDD, « Resilient Distributed Dataset », est une abstraction de collection sur laquelle est réalisée des opérations de manières distribuées
- Programmation différente d'Hadoop



Etat de l'art – Pig & Hive

- Infrastructure construite par-dessus Hadoop
- Utilise PigLatin, un langage de script
- Hive, permet de réaliser des requêtes en HiveQL, un langage proche du SQL
- Très bon outil pour développer de rapide job MapReduce



Découverte et expérimentation

- Objectif prendre en main le framework Hadoop
- Choix de la distribution HortonWorks
- Installation d'une distribution en local avec Eclipse



Hortonworks Sandbox with HDP 2.2

[Leave Feedback](#)

Component	Version
Map	2.6.1.0
HDP	2.2.4
Hadoop	2.6.0
Pig	0.14.0
Hive-Metastore	0.14.0
Cloudera	4.1.0
Ambari	2.5.10 Details
Yarn	2.6.0
Tez	0.8.4
Storm	0.9.3
Falcon	0.8.0
Sandbox Build	114638 (0-25 03-04-15)

My scripts

+ new script

1-2 Overview

Settings

Email notification

+ add another notification

+ add another ID or

Title

My script #1

```

Cycle = LOAD 'default/cycle' USING org.apache.hadoop.mapreduce.lib.input.TextInputFormat AS (file:double, de:double, mag:double, hp: int);
MagMax = LOAD 'default/magmax' USING org.apache.hadoop.mapreduce.lib.input.TextInputFormat AS (hp: int, mag:double, de:double, file:double);
Joiner = JOIN Cycle BY hp, MagMax BY hp;
Order = ORDER Joiner BY hp asc;
Result = FILTER Order BY file = 42000;
DUMP result;

```

```

Cycle = LOAD 'default/cycle' USING org.apache.hadoop.mapreduce.lib.input.TextInputFormat AS (file:double, de:double, mag:double, hp: int);
MagMax = LOAD 'default/magmax' USING
org.apache.hadoop.mapreduce.lib.input.TextInputFormat AS (hp: int, mag:double, de:double, file:double);
Joiner = JOIN Cycle BY hp, MagMax BY hp;
Order = ORDER Joiner BY hp asc;
Result = FILTER Order BY file = 42000;
DUMP result;

```

Settings 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

Run Cancel Stop Refresh

1

Découverte et expérimentation -Hive

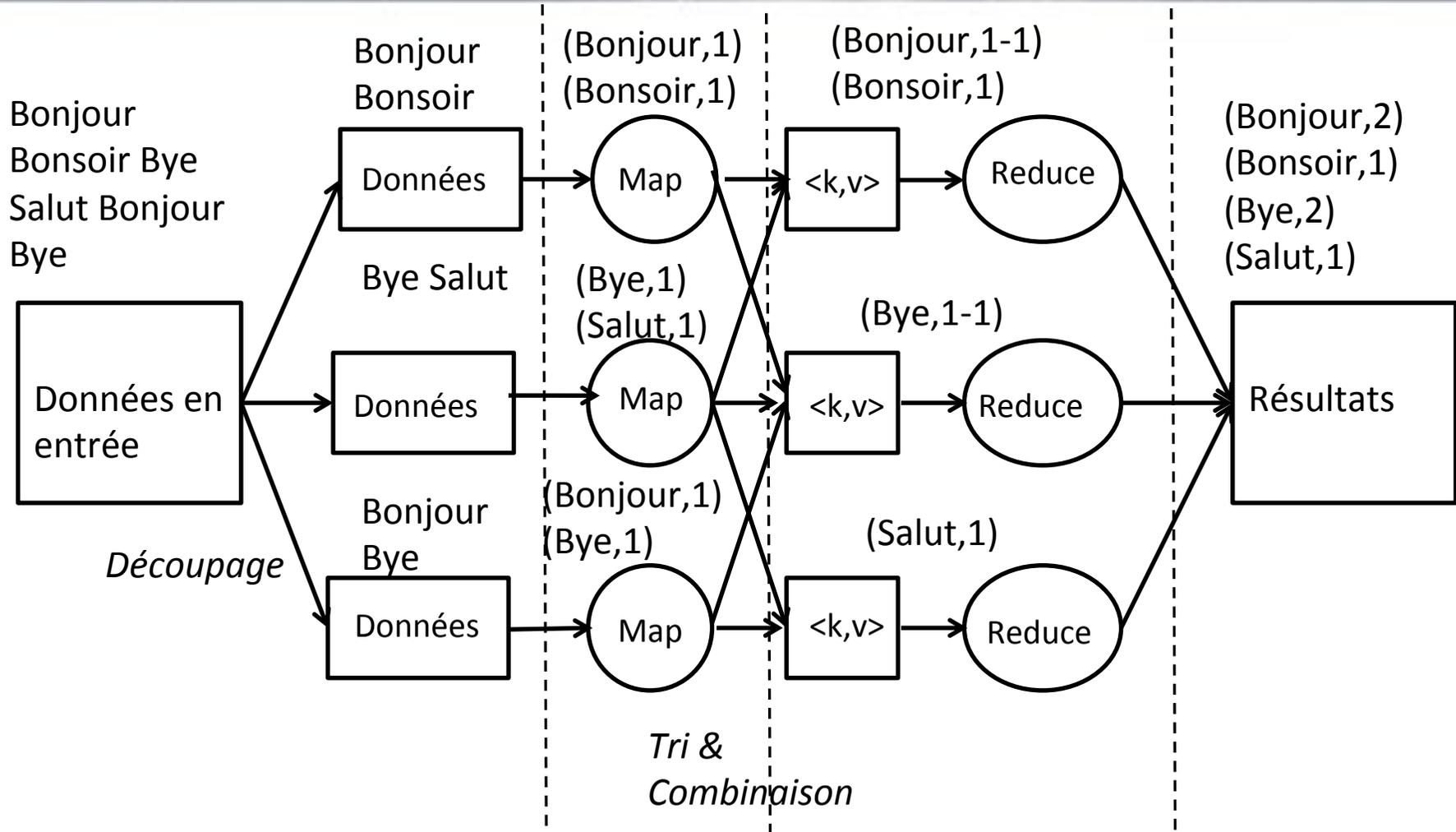
- Simplification d'un job MapReduce
- Utilisation d'un langage HiveQL très proche du SQL

Bilan sur Hive et Pig

- Accessible pour des programmeurs non formés à Hadoop
- Inconvénients :
 - Presque inefficace pour des programmes trop complexes
 - Ne supporte pas tous les standards attendus

Découverte et expérimentation – Essais avec Hadoop

- Utilisation de Maven pour créer les projets
- Programmeur doit coder : une fonction *map()*, une fonction *reduce()*, et un driver
- Les données chargées sont découpées sous forme de bloc puis distribuées sur le cluster



Découverte et expérimentation – Essais avec Spark

- Chargement des données dans des javaRDD
- Utilisation de méthode emprunter au langage fonctionnel pour réaliser des traitements sur les JavaRDD
- Utilisation des classes Function, Function2,... permettant de définir les types d'entrées et de sorties

```

class PairRDD[String, String] extends RDD[String] with Serializable {
  private val f: Function2[String, String, String] = ...

  public Tuple2[String, String] call(String a) {
    // Je découpe la ligne pour récupérer chaque des valeurs
    String[] t = a.split(",");
    // Je stocke la valeur ra et de et je les converti en double
    Double ra = Double.parseDouble(t[0]);
    Double de = Double.parseDouble(t[1]);
    // Je calcule la valeur de l'ha
    Long haid = height(ra, de);
    return new Tuple2[String, String](haid, callString(a));
  }
}

```

Application d'une fonction, dont le paramètre est une classe dont on peut définir les types d'entrées et de sorties, puis application sur chaque élément de notre JavaRDD d'un traitement correspondant à la fonction call contenu dans la classe interne.

Application

- Détermination d'une application à réaliser pour l'Observatoire
- Réalisation d'une application de CrossMatch

Application - CrossMatch

- Réalisation d'une jointure entre 2 catalogues
- Calcul d'un identifiant servant de clé de jointure
- Filtration des sources en calculant la distance entre elles

Application - Description des sources

- 2 Catalogues :
 - Tycho : 2 539 913 valeurs
 - HipMain : 119 218 valeurs

```
45.084243,+0.260814,10.723,-1  
45.084843,+0.260803,10.977,-1  
45.104016,+0.300000,10.500,-1  
45.140714,+0.300004,10.304,-1  
45.182794,+0.300000,10.100,-1  
45.112794,+0.300014,10.204,-1  
45.010100,+0.301004,10.100,-1  
45.004000,+0.470000,9.970,-1  
45.074004,+0.470007,10.300,-1
```

```
3, 9.27, 0.00379707, -29.48882768, 21.9  
3, 8.41, 0.00000799, 28.80018000, 3.82  
4, 9.04, 0.00000007, -41.89914432, 7.76  
5, 9.56, 0.00000004, -49.85120448, 2.87  
6, 12.01, 0.01014144, 3.74048830, 28.8  
7, 9.66, 0.00204001, 10.00000000, 27.76  
8, 9.04, 0.02000000, 28.80018000, 8.17  
9, 9.04, 0.00000000, 49.80000000, 4.80  
10, 8.00, 0.00000000, -50.00000000, 10.74
```

Application – Réalisation avec Hadoop

- Réalisation de recherches pour trouver une façon de faire une jointure
- Construction du code :
 - 2 Mappers, un par catalogue
 - 1 Reducer
 - 1 Driver

Application - HipMainJoinMapper

- Découpage de la ligne courante pour obtenir valeurs de ra et de
- Calcul de l'identifiant à partir de ces 2 valeurs
- Création du couple <clé,valeur> :
 - Clé -> l'identifiant
 - valeur -> la ligne courante + une lettre

```
@Override
public void map(Object key, Object value, OutputCollector context,
    Reporter arg) throws IOException {
    String[] parsed = value.toString().split(",");
    if(parsed.length==4){
        String ra = parsed[2];
        String dec = parsed[3];
        double raDeg = Double.parseDouble(ra);
        double decDeg = Double.parseDouble(dec);
        //calcul de l'id
        long id = nextId(raDeg,decDeg);
        //on passe l'id comme clé du couple
        outkey.set(id+"");
        //on passe la valeur de la ligne = à pour désigner la src d'origine
        outvalue.set("R" + value.toString());
        //on passe le couple valeur en sortie
        context.collect(outkey,outvalue);

        // traite les pixels voisins
    }
    try {
        for (long nei : nb.neighbours(4)) {
            outkey.set(nei+"");
            context.collect(outkey,outvalue);
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Application - CrossMatchReducer

- En entrées : un couple <clé,Collection<valeur>>
- Ces valeurs viennent des 2 catalogues, identification par la lettre
- Parcours de la Collection<valeur> pour ranger chaque valeur dans la liste correspondante
- Réalisation de la jointure en parcourant les 2 listes, filtration des données avant écriture

```
private void executeJoinLogic(OutputCollector context) throws IOException {
    if(!listA.isEmpty() && !listB.isEmpty()){
        for(Text a : listA){
            // EXTRAITE POS A
            String[] a = a.toString().split(",");

            double val = Double.parseDouble(a[0]);
            double dec1 = Double.parseDouble(a[1]);

            for(Text b : listB) {
                // EXTRAITE POS B
                String[] b = b.toString().split(",");
                double val2 = Double.parseDouble(b[0]);
                double dec2 = Double.parseDouble(b[1]);
                double distance = haversine(val,dec1,val2,dec2);
                if (distance < matchDistance) {
                    context.collect(a,b);
                }
            }
        }
    }
}

@Override
public void reduce(Object key, Iterator values, OutputCollector context,
    Reporter arg) throws IOException {
    listA.clear();
    listB.clear();
    while(values.hasNext()){
        key = (Text) values.next();
        if(key.charAt(0) == 'A'){
            listA.add(new Text(key.toString().substring(1)));
        }else if(key.charAt(0) == 'B'){
            listB.add(new Text(key.toString().substring(1)));
        }
    }
    try {
        executeJoinLogic(context);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Application - Résultat

- En entrée on a donc :
 - Tycho : 2,5 Millions valeurs
 - HipMain : 120 000 valeurs
- En sortie : 119 167 valeurs

Conclusion

- Une expérience enrichissante
- Difficultés rencontrées
- Conclusion sur Hadoop et Spark



Merci de m'avoir écouté, avez-vous
des questions ?