Spark-ICS

Exploration de l'application de l'architecture Apache Spark à la bioinformatique

Axel Verdier, Ludovic Legrand, Xavier Garnier, Erika Sallet, Alexandre Dehne-Garcia, Nicolas Lapalu, Martial Briand, Franck Dorkeld, Bernhard Gschloessl, Corinne Rancurel, Martine Da Rocha, Sébastien Carrère, Céline Noirot, Olivier Filangi, Jérôme Gouzy

SUNRISE
UNE CULTURE POUR LE FUTUR

Laboratoire Interactions Plantes Micro-organismes

Le projet SPARK-ICS vise à évaluer une solution technologique du « Big Data » pour résoudre des classes de problèmes bioinformatiques qui atteignent la limite des architectures actuelles.

La quantité et la diversité des données ne font que croître grâce à l'amélioration des débits et la baisse des coûts induisant des problèmes de passage à l'échelle et augmentant la complexité des analyses. En outre, une meilleure compréhension des jeux de données complexes nécessite de plus en plus l'utilisation du Machine Learning pour capter l'information la plus pertinente.

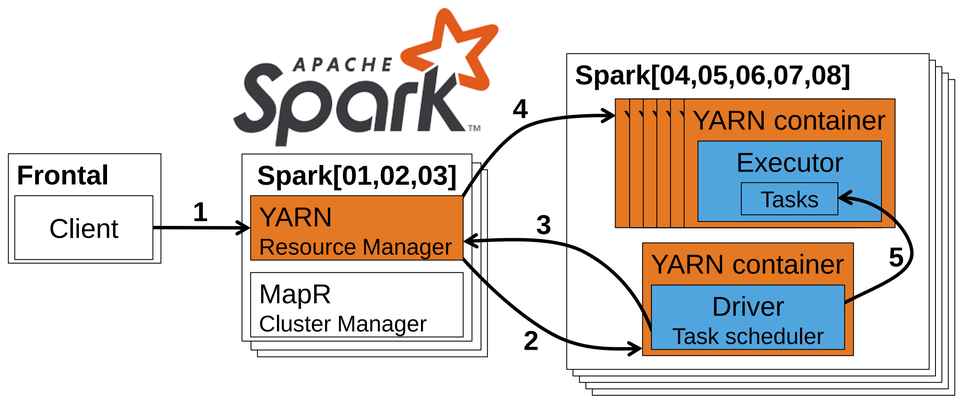
L'architecture Apache Spark intègre une grande variété d'outils et de méthodes du Machine Learning dans un environnement de calcul pouvant être instancié sur un poste de travail, un cluster de calcul ou dans le cloud. Depuis deux ans notre groupe de travail a évalué cette architecture pour répondre à des questions de bioinformatique.

Objectifs

- Evaluer Spark dans le cadre de la bioinformatique
  - Evaluer des outils existants
  - Développer de nouveaux outils
- Accélérer et améliorer l'analyse des données
- Acquérir des compétences en « Big Data » avec Spark
  - Développement en Scala
  - Mise en place d'un cluster
  - Machine Learning

Fonctionnement de Spark

1. Soumission d'un job par le client avec X exécuteurs. YARN valide ou met en attente le job en fonction des ressources disponibles
2. Instanciation d'un conteneur et d'un driver Spark
3. Construction d'un graphe de tâches et demande de ressources à YARN
4. YARN instancie des conteneurs et le driver démarre les exécuteurs dans les conteneurs
5. Le driver répartit les tâches sur les exécuteurs et gère les erreurs

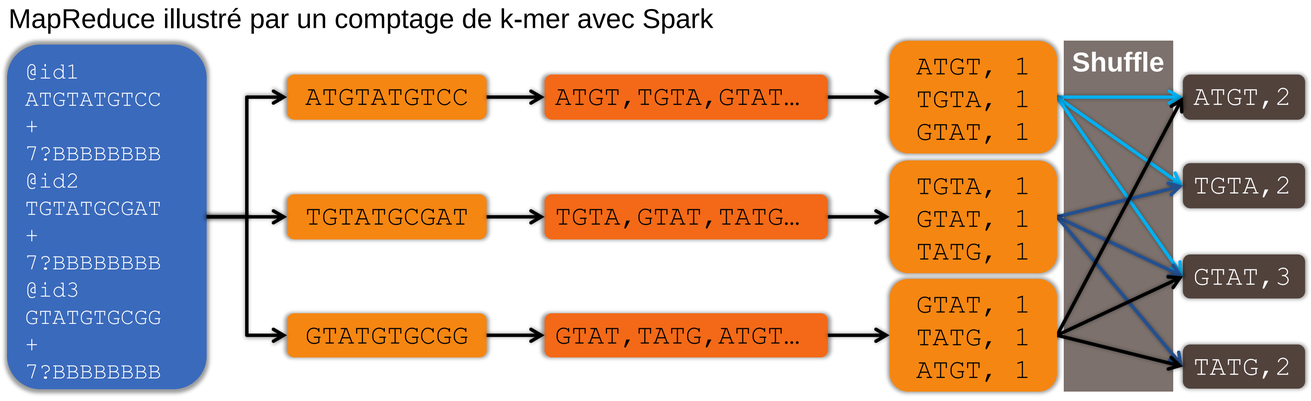


Cluster Spark

- Cluster de 8 nœuds physiques
  - 252 cœurs disponibles
  - 1,4To de RAM utilisables
  - 34To de disque brut
- 4 nœuds avec 386Go de RAM et CPU à 2,4GHz
- 4 nœuds avec 256Go de RAM et CPU à 2,2GHz
- Un serveur frontal pour soumettre les jobs

Note: le même cluster est aussi accessible via le scheduler SGE

MapReduce illustré par un comptage de k-mer avec Spark

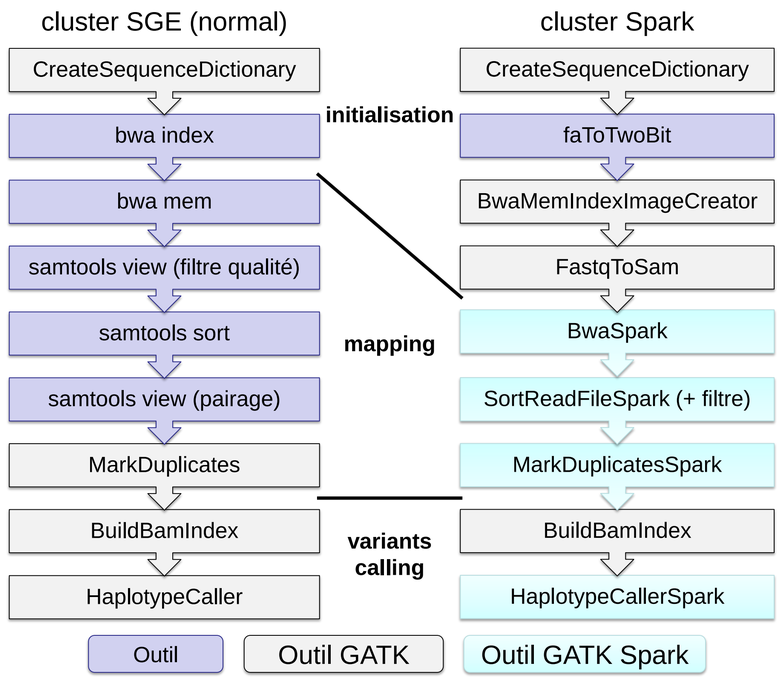



```
sc.textFile("data.fastq").sliding(4,4).map{ case Array(_,seq,_,_) => seq }.flatMap( read => read.sliding(4,1) ).map( kmer => (kmer,1) ).reduceByKey( (accu,n) => (accu + n) )
```

MapReduce[1] avec Spark

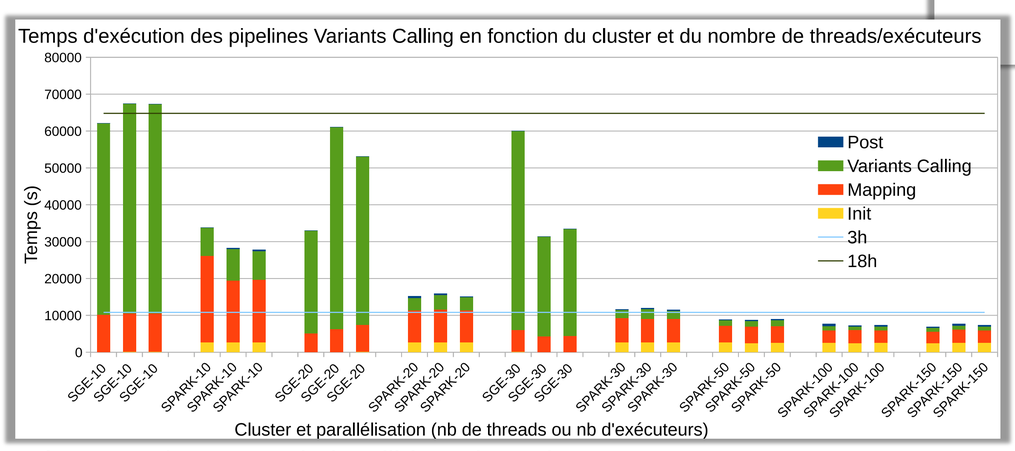
- Jusqu'à **100x plus performant que MapReduce avec Hadoop**
  - Utilisation de la **RAM** au lieu des disques entre les tâches
  - Réorganisation et optimisation des tâches avant l'exécution
  - Une API plus riche et plus généraliste
- Distribution et reprise sur erreur gérées par Spark
- Le **Shuffle** qui implique du trafic réseau et de l'écriture sur disque, reste une étape critique pour les performances

Benchmarking

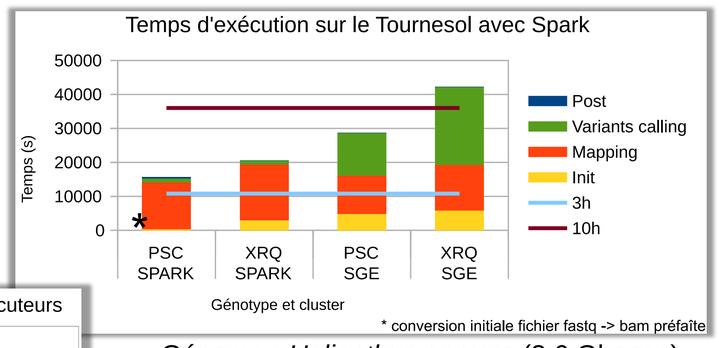


Pipeline GATK4[2]

- Outil pour la détection de variants (v4.0.1.2)
- Utilisation des outils **HaplotypeCaller** et **HaplotypeCallerSpark**
- Les versions non-Spark et Spark ont un algorithme différent
- Benchmark SGE et Spark sur les mêmes serveurs



Génome : *Plasmopara halstedii* (75 Mbases), fastq.gz : 2x12Go, 3 répétitions



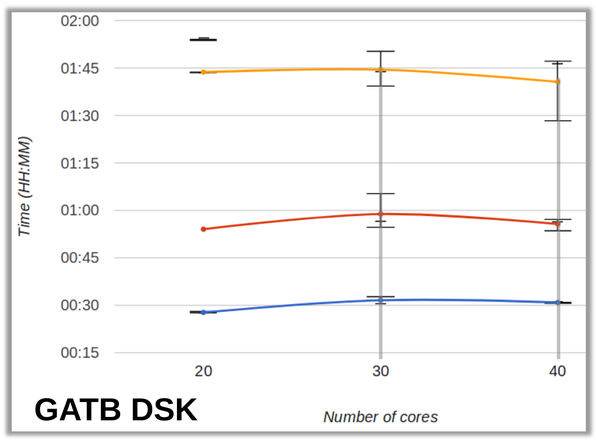
Génome : *Helianthus annuus* (3,6 Gbases)
2 génotypes : PSC et XRQ
fastq.gz : 2x12Go, sans répétition

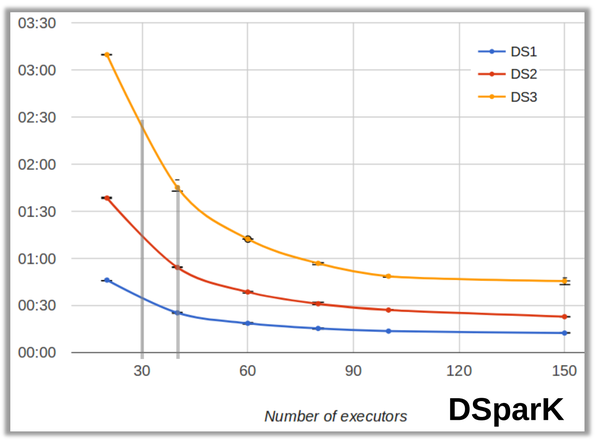
- La version non-Spark est peu parallélisée
- La version Spark est encore en bêta
- **Gain de temps** conséquent avec la version Spark

Réingénierie en Scala/Spark d'outils bioinformatiques

DSparK

- Porter un outil C++ pour le comptage de k-mer
  - DSK fait partie de **GATB-Core**[3]
- Jeux de données (format fasta)
  - DS1 : 800 millions de reads
  - DS2 : 1,6 milliards de reads
  - DS3 : 3,2 milliards de reads





- Spark est moins performant que le C++ à ressources équivalentes
- Spark permet un **meilleur passage à l'échelle**
- Développement en Spark plus rapide

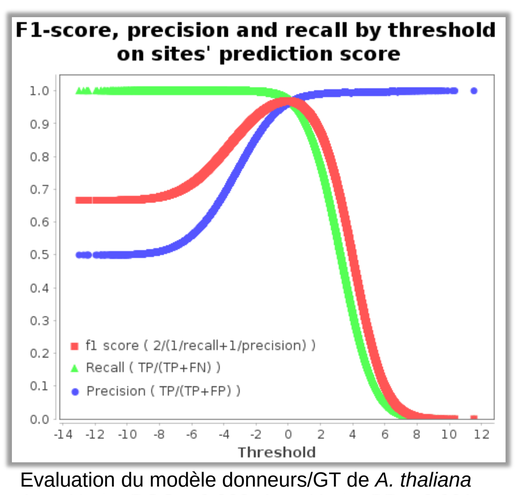
SpliceMachine[4]

- **Objectif** : Détection des sites d'épissage par utilisation de Linear Support Vector Machine
- Développement de l'outil SpliceMachine sous Spark

- **Entraînement en quelques heures** contre plusieurs jours pour l'ancienne version
- Spark permet une **implémentation simple** du **Machine Learning**, quelques dizaines de lignes Scala pour entrainer le modèle

Entraînement du modèle

- Extraction des features dans une fenêtre de +/- 70nt autour des sites donneurs (GT) ou accepteurs (AG)
- **22608 features** en 3 classes :
  - 11472 **Positional Information** : composition en 1 à 3-mers aux positions
  - 10752 **Compositional Information** : présence/absence des 4 à 6-mers
  - 384 **Coding Potential** : Présence des codons pour chacun des 3 cadres de lecture.
- Construction d'un jeu de 87454 sites positifs et 87454 sites négatifs à partir de données de transcrits *A. thaliana*



Evaluation du modèle donneurs/GT de *A. thaliana*
Area Under ROC = 0,993; Area Under PR = 0,991

[1] Zaharia, M. et al., 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. NSDI. https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf
[2] McKenna, A. et al., 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. GENOME RESEARCH 20:1297-303
[3] Genome Analysis Toolbox with de-Bruijn graph (GATB). https://gatb.inria.fr/
[4] Degroeve, S. et al., 2005. SpliceMachine: Predicting splice sites from high-dimensional local context representations. Bioinformatics, 21(8), 1332–1338.