Chapitre 1

Définitions et objectifs du cours

Ce cours d'informatique pour les données et calculs massifs, présente des concepts issus de deux grands domaines scientifiques et techniques : la *Science des données et des Big Data* d'une part, et le *calcul parallèle, ou High Performance Computing (HPC)*, d'autre part. Afin de préciser le cadre du cours et de lever d'éventuelles confusions, ce chapitre introductif résume ce que sont ces deux grands domaines et liste les objectifs du cours.

1.1 Définition du *Big Data*

Big Data ou *Data Science* ?

Certains considèrent que l'analyse des données et l'apprentissage numérique incluent forcément les méthodes et technologies pour traiter de très gros volumes de données, et font ainsi du *Big Data* une partie des *Data Sciences*. A l'opposé, d'autres considèrent que le traitement de très gros volumes de données inclut forcément des méthodes d'analyse probabiliste et d'apprentissage, et considèrent que les *Data Sciences* sont inclues dans le *Big Data*. Ce cours aborde autant les problématiques d'exploitation de très gros volumes de données que celles de leur analyse, mais il reste un cours **d'informatique**. Le titre de *Big Data* a donc été préféré à celui de *Data Sciences*, et les aspects mathématiques des algorithmes d'analyse de données seront très peu abordés.

Disciplines participant au Big Data

Le Big Data est un domaine pluridisciplinaire pour lequel on peut identifier 5 parties, parfois elles-mêmes basées sur plusieurs disciplines (voir figure 1.1). On peut tout d'abord énumérer quatre parties clés :

- Une partie qualifiée parfois de *Math-Info* comprend tout d'abord des *mathématiques statistiques et probabilistes* sur lesquelles sont fondés des *algorithmes d'apprentissage numérique* (ou *machine learning*), ainsi que des algorithmes de fouille de données et de graphes. Cette partie du Big Data est celle qui est souvent identifiée comme le *Data Science*. C'est en tous cas le cœur mathématique du Big Data.

- Une partie *d'informatique distribué pour l'analyse de données large échelle*. Il s'agit d'une forme d'algorithmique distribuée récente (apparue en 2009), visant à amener les traitements sur les machines où sont stockées les données. Cette approche permet des traitements de données à large échelle (sur des données très volumineuses), voire à l'échelle du web (*web-scale*). Une première mise en œuvre de cette approche utilisait le schéma *Map-Reduce* : un

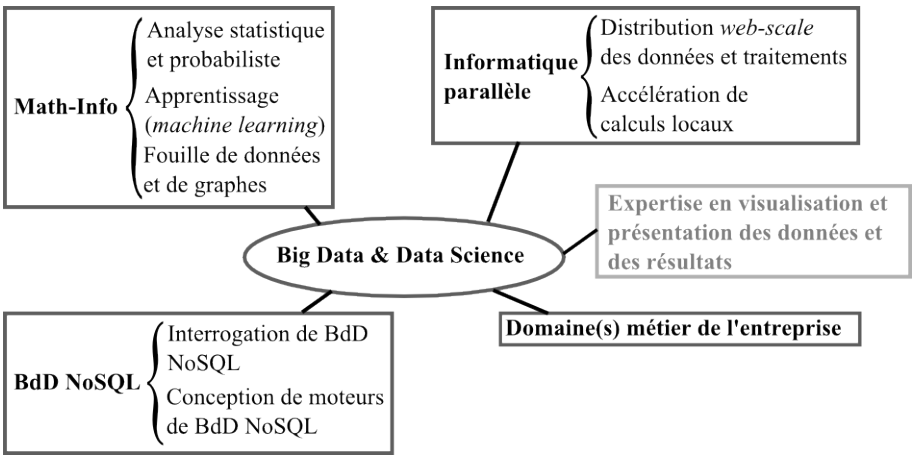

FIGURE 1.1 – Composition pluridisciplinaire du *Big Data*

schéma de calcul distribué à première vue très contraint mais en fait assez générique.

– Une partie *d'informatique parallèle à haute performance pour le data analytics et le machine learning* visant à accélérer les calculs sur des machines parallèles. Par exemple, en utilisant un cluster de PC multi-cœurs (ensemble de PC dédiés aux calculs et reliés par un réseau local rapide) ou un cluster de GPU (réseau de cartes graphiques détournées pour du calcul scientifique), pour entraîner des réseaux de neurones profonds (*deep learning*).

– Une autre partie essentielle du Big Data réside dans la conception et *l'exploitation de bases de données "not only SQL" (NoSQL)*. Elles permettent de stocker des données structurées complexes, ou au contraire de simples fichiers textes que l'on devra analyser en détail. Certaines BdD NoSQL ont été conçues pour un stockage distribué à très large échelle, d'autres pour favoriser la vitesse d'interrogation sur des données plus restreintes. Le domaine des BdD NoSQL est encore en pleine évolution.

Deux autres parties complètent l'aspect pluridisciplinaire du Big Data : le domaine applicatif considéré, et la visualisation et présentation des données et résultats.

– Une connaissance du *domaine d'activité de l'entreprise* est nécessaire pour que le *data scientist* puisse donner du sens aux données, guider son analyse et interpréter les résultats de ses algorithmes.

– Enfin, le *data scientist* doit aussi posséder une expertise en *visualisation de gros volumes de données* et en *présentation synthétique/simplifiée des résultats*. Cette facette de ces compétences et activités est essentielle pour aboutir à une prise de décision dans un contexte industriel.

Exemples d'applications Big Data

Parmi les applications du Big Data on peut citer trois grands classiques. Premièrement l'analyse de fichiers de traces pour en déduire des comportements passés et futurs : traces de transactions commerciales, traces d'activités de personnes mobiles, traces d'accès à des serveurs web…, et déductions de comportements de consommateurs, de voyageurs, d'utilisation d'Internet… Cette analyse aboutit généralement à un système de recommandation pour mieux cibler des clients po-

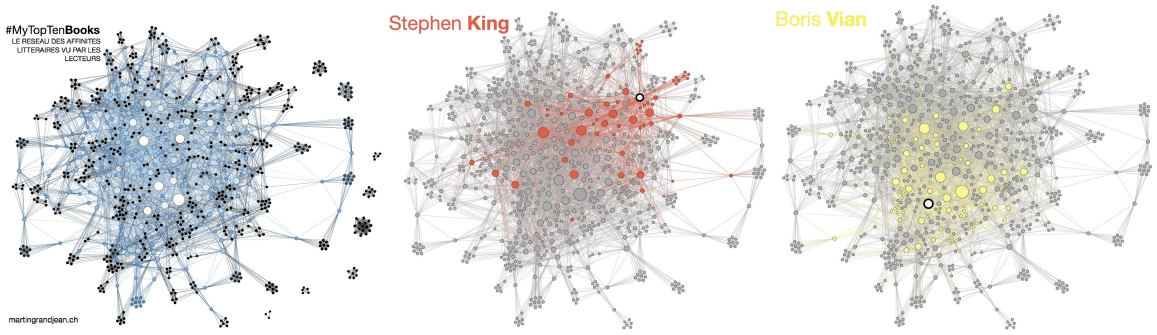

FIGURE 1.2 – Graphe des auteurs littéraires dont les œuvres apparaissent dans des *Top10* de lecteurs : deux auteurs sont reliés s'ils sont présent dans une même *Top10* (expérience *#MyTopTenBooks*, voir `http://www.martingrandjean.ch/`). On observe que les graphes des auteurs connectés à Stephen King et à Boris Vian partagent quelques nœuds, certains auteurs sont donc appréciés à la fois des lecteurs de Stephen King et de ceux de Boris Vian.

tentiels ou pour augmenter les performances de l'entreprise. Ce type d'analyse se faisait initialement *off line* sur de très gros volumes de données, mais on observe une demande croissante de traitement *on line* de flux continus de données.

Deuxièmement, l'analyse de signaux de sorties d'une foule de capteurs sur une installation industrielle, couplée à l'accès à des bases de données de situations passées et de spécifications techniques, afin d'identifier les prémisses d'une défaillance future. Cette analyse doit en général se faire en temps réel à partir des signaux captés sur l'installation en cours d'utilisation.

Troisièmement, l'analyse de réseaux sociaux, c'est-à-dire de l'analyse de graphes, afin d'en déduire par exemple des relations/influences entre individus ou populations d'individus. L'analyse de graphes de grandes tailles constitue une partie spécifique du Big Data souvent citée comme exemple, et qui intéresse particulièrement quelques *géants du web*. La figure 1.2 donne l'exemple d'un graphe d'auteurs littéraires partageant des *Top10* exprimés par des lecteurs. En analysant ce graphe avec des outils adaptés on peut facilement visualiser les auteurs qui plaisent aussi aux lecteurs de Stephen King, ou à ceux de Boris Vian, ou aux lecteurs des deux.

Du *statisticien* au *data scientist*

Comparé au traditionnel statisticien du passé dans l'entreprise, le *data scientist* est probablement moins pointu en mathématique mais pratique la pluridisciplinarité. Il possède un niveau de développement informatique lui permettant de prototyper et tester facilement les algorithmes d'analyse de données qu'il conçoit. De plus, il s'implique dans le métier premier de l'entreprise afin de savoir mener des analyses de données pertinentes pour, par exemple, identifier de nouveaux marchés.

Enfin, une partie du travail du *data scientist* consiste à simplifier, synthétiser et présenter les résultats de ses analyses aux autres membres de l'entreprises. Cette dernière facette de son travail est primordiale car il doit convaincre ses collègues de la pertinence et de la solidité de ses conclusions sur l'évolution d'un process industriel ou de la stratégie de l'entreprise.

Le *data engineer*

Les couches logicielles d'un système *Big Data* sont devenues très complexes. Stocker et mettre à jour en permanence un *Data Lake* demande beaucoup de ressources matérielles, mais aussi une architecture logicielle efficace et tolérante aux pannes, qui dépasse de loin les simples serveurs

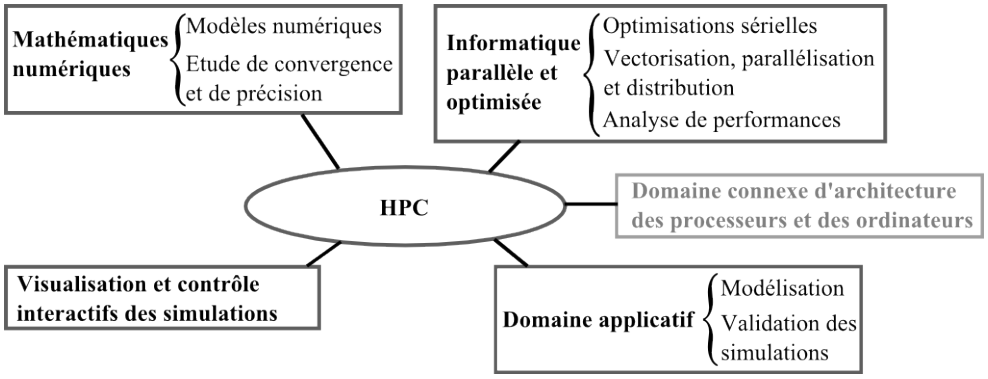

FIGURE 1.3 – Composition pluridisciplinaire du *HPC*

de comptes et de données installés sur un petit réseau local. Un interfaçage avec des sources de données externes, ou avec des flux de données continus est aussi devenu une composante classique d'un système *Big Data*. De plus, analyser de gros volumes de données sur ces architectures logicielles et matérielles requiert également une expérience spécifique en algorithmique et en programmation, pour aboutir à des solutions efficaces à large échelle. Enfin, avant d'analyser des *masses de données* il convient de *collecter*, *nettoyer* et *enrichir* ces données, qui sont souvent hétérogènes, redondantes, mal identifiées…

Toutes ces tâches (et d'autres) constituent des travaux d'ingénieurs apparus avec les *Big Data*, et définissent le travail des *Data Engineers*. Il est habituel que 80% d'un projet *Big Data* soit constitué de *Data Engineering* (et de seulement 20% de *Data Science*).

Langages de programmation actuels du Big Data

Les langages interprétés comme Java et Python, qui sont parmi les plus portables, sont les langages de développement de bas niveau du Big Data. Ce qui n'empêche pas certaines bibliothèques d'apprentissage numérique d'être développées en C ou C++. Mais le *data scientist* se limite habituellement à des développement en Java ou Python, qu'il interface avec des bibliothèques éventuellement développées dans d'autres environnements.

Enfin, les langages de requêtes des BdD NoSQL constituent souvent les langages de haut niveau du Big Data, et s'interfacent aussi avec des codes Java et Python.

1.2 Définition du *calcul à haute performance (HPC)*

Disciplines participant au HPC

Le HPC est un autre domaine pluridisciplinaire. Il inclut de l'informatique parallèle, des mathématiques numériques, des domaines applicatifs et de la visualisation interactive (voir figure 1.3) :

– Le *domaine applicatif* fournit des problèmes qui nécessitent d'être modélisés et simulés avec précision, et qui sont souvent de grandes tailles. Une expertise de la physique du domaine permet d'en établir un modèle mathématique avec la précision requise.

– Les *mathématiques numériques* permettent ensuite de transformer ce modèle en un algorithme numérique convergeant vers le bon résultat et conservant la précision voulue. La

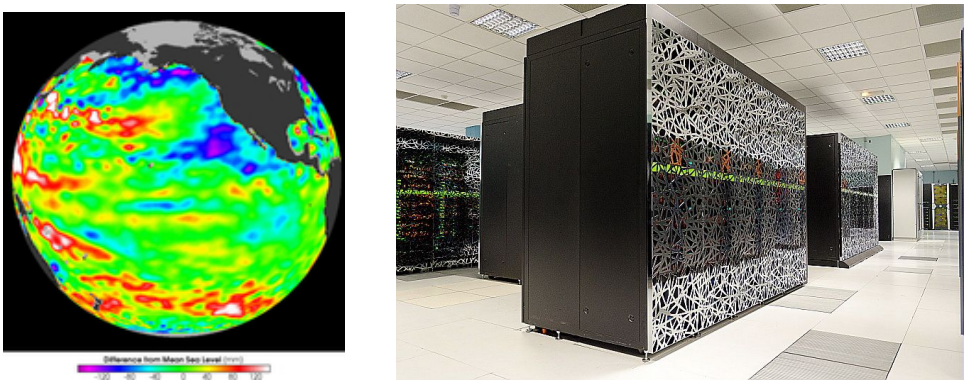

FIGURE 1.4 – Exemple de simulation météo globale pour la mise en évidence *d'el niño* par l'INRIA, et exemple de supercalculateur BullX de Meteo France

maîtrise des problèmes d'arrondis fait partie de cette expertise.

– Puis, *l'informatique parallèle et l'optimisation de code* permettent de transformer un algorithme numérique en un algorithme parallèle puis en un code optimisé pour l'architecture informatique visée. L'utilisation optimale de toutes les unités de calculs disponibles est au cœur de cette dernière étape, et une expertise en analyse de performances est également nécessaire pour y arriver.

– D'autre part, le HPC nécessite de concevoir des logiciels permettant une *visualisation et un contrôle interactifs* des simulations par plusieurs experts à la fois. Cette composante du HPC a pris progressivement une grande importance.

Enfin, une expertise dans le domaine connexe de *l'architecture des processeurs et des ordinateurs* est nécessaire pour la conception et l'implantation de codes à haute performance.

L'*intrication* : la spécificité des codes HPC

La clé du succès d'une démarche HPC est *l'intrication* de toutes les étapes du développement. L'informatique traditionnelle raisonne par *couches* conçues séparément, respectant des interfaces les plus génériques possibles, et sans soucis de l'architecture finalement utilisée. A l'opposé, le HPC vise à concevoir un modèle mathématique, puis un algorithme numérique et enfin un code exécutable qui visent tous à utiliser au mieux les capacités de la machine cible (ou d'une famille de machines cibles). Toutes les étapes du développement deviennent alors interdépendantes et *intriquées*.

Exemples d'applications HPC

Aujourd'hui il existe de très nombreux domaines d'applications du HPC, impossibles à tous énumérer. Les calculs de météo sont l'une des applications du HPC les plus utiles à chacun d'entre nous. Comparé à d'autres applications, les codes de calcul de météo ont besoin d'accéder à d'importants volumes de données (les relevés de température dans le monde, par exemple), ce qui pose de nombreux problèmes d'optimisation de ces codes sur des supercalculateurs (voir figure 1.4).

Les calculs de neutroniques concernent un public beaucoup plus restreint, mais sont un pilier essentiel du HPC. Ils sont indispensables à la mise au point des réacteurs nucléaires. Ces codes

ont des cycles de vie très longs et ont été très fortement optimisés pour les architectures parallèles successives.

Les *problèmes inverses*, comme les diverses techniques de tomographie qui consiste à reconstituer une image 3D d'un *corps* en fonction de diverses projections de rayons à travers ce corps, sont très gourmands en temps de calcul. Or, il est parfois souhaitable de disposer des résultats rapidement, voire en temps réel. Les calculs de problèmes inverses exploitent généralement des architectures parallèles et leurs codes sont développés suivant une démarche HPC.

Technologies traditionnelles du HPC

Les codes HPC sont complexes, et longs à concevoir et à mettre au point. Leur cycle de vie peut atteindre 30 ans, avec des évolutions régulières fonctions des progrès des architectures informatiques et des méthodes numériques. Les implantations se font en langages compilés (plus rapides) comme le FORTRAN, le C ou plus récemment le C++. Les outils de parallélisation les plus standards sont OpenMP pour le développement multithreads (au sein de PC multi-cœurs), MPI pour le développement distribué (sur cluster de PCs), et CUDA et OpenACC pour le développement sur GPU.

Certains calculs mathématiques sont présents dans de très nombreux codes de calculs. Afin de capitaliser les *hommes* $\times$ *ans* de travail algorithmique et d'implantation associés à ces calculs, de nombreuses bibliothèques de calculs HPC ont vu le jour. Les *BLAS* implantent ainsi des calculs d'algèbre linéaire (calculs sur des vecteurs et des matrices). Leur interface est entièrement normalisée, et il en existe plusieurs implantations. Les plus connues sont très optimisées et si un développeur non spécialiste décide de les ré-implanter, il n'atteindra au final qu'une fraction de leur performance.

1.3 Participation du HPC au Big Data

Les premiers développements du Big Data ont été réalisés principalement dans le but d'offrir des fonctionnalités manquantes pour analyser facilement d'énormes volumes de données distribuées à large échelle. L'idée clé était d'amener une partie des calculs sur les sites où se trouvaient les données, plutôt que de rassembler les données sur un seul site pour les traiter ensuite sur une seule machine (ex : application du paradigme *Map-Reduce* dans l'environnement *Hadoop*). Transmettre quelques codes de calculs sur Internet et collecter des résultats pré-agrégés semble logiquement plus rapide que de transmettre d'énormes volumes de données brutes pour les traiter ensuite sur un seul site.

A l'opposé, la communauté HPC (académique et industrielle) traite de gros volumes de données, mais qui sont stockés localement sur des baies disques rapides reliées très efficacement à des baies de calculs. De plus les données sont habituellement dans un format binaire et adapté aux calculs réalisés. On parle parfois *d'Extreme Data*, comme lors d'une simulation de l'univers entier (réalisée en France sur le supercalculateur Curie) et qui a nécessité environ 640To de mémoire RAM.

Initialement la communauté HPC ne pratiquait donc pas d'analyse de données *web-scale* au sens de la communauté Big Data. Cependant, des soucis de performances dans les applications Big Data, ainsi que l'accroissement des enjeux industriels du Big Data ont motivé l'utilisation de technologies et d'expertises HPC pour le Big Data :

– *Hadoop* est un célèbre environnement de développement et d'exécution distribuée d'analyse de données à large échelle. Il est basé sur le paradigme *Map-Reduce*, et est écrit et programmable en Java. *Hadoop* a prouvé sa généricité pour de l'analyse de données, néanmoins il se

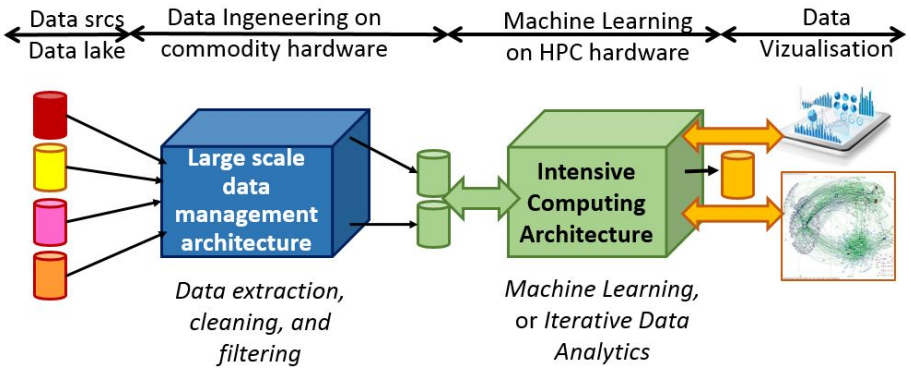

FIGURE 1.5 – Architecture mixte d'analyse de données à large échelle et à haute performance

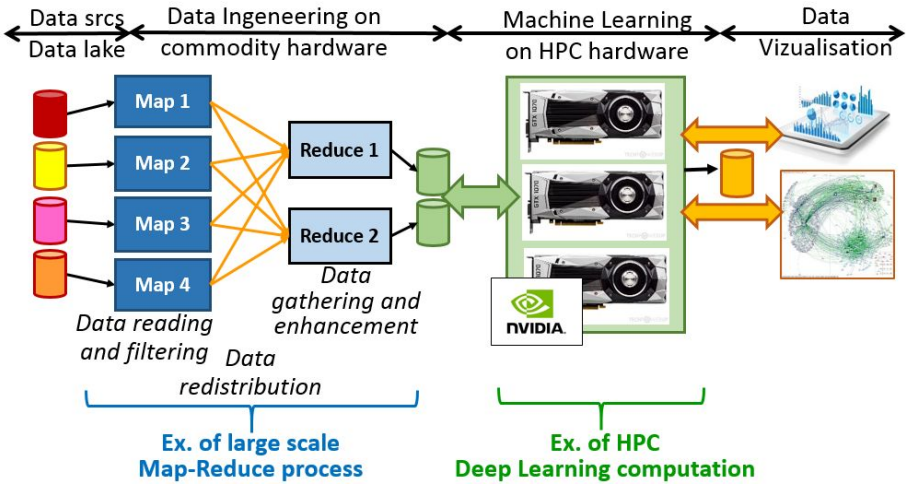

FIGURE 1.6 – Exemple de mise en œuvre d'une analyse de données à large échelle et à haute performance

repose beaucoup sur la lecture et l'écriture de fichiers (données initiales et résultats finaux, mais aussi résultats intermédiaires). On a alors vu apparaître Spark, un autre environnement de développement *Map-Reduce* qui organise ses calculs en mémoire et fait moins souvent appel au système de fichiers. Spark est considéré plus performant qu'*Hadoop*, et partage la démarche de calcul *in memory* du HPC.

– Certaines BdD *NoSQL* ayant des objectifs de performance ou de très large échelle ont été initialement développées en Java, et ont finalement vu certains de leurs composants critiques être réimplanté en C++ pour améliorer leurs performances. D'autres one été développés dès le départ en C++, comme *MongoDB* (voir section 9.3).

– Des algorithmes d'apprentissage ont été re-développés pour des architectures parallèles (y compris sur des GPU [1]), et ont subi des optimisations très poussées. C'est le cas, par exemple, des algorithmes de *Deep Learning* et de *DBSCAN*.

Aujourd'hui la rencontre des technologies et du savoir-faire HPC avec les besoins du Big Data donne naissance à ce qui est parfois appelé le *High Performance Data Analytics* (**HPDA**),

1. processeurs graphiques utilisés ici en tant que co-processeurs de calcul scientifique

afin de réaliser de l'analyse de données complexe à grande échelle (et donc le plus efficacement possible). On déploie alors des architectures matérielles et logicielles mixtes *"analyse de données & calcul intensif"*. La figure 1.5 en donne un exemple avec une partie *large échelle* de filtrage, nettoyage et enrichissement de données provenant de multiples sources (partie gauche), puis une partie *HPC* d'apprentissage automatique sur les données filtrées et enrichies. Une dernière partie de visualisation confortable des résultats, permettant leur présentation aux différents acteurs de l'entreprise, termine cette chaîne d'analyse de données.

La figure 1.6 décrit une mise en œuvre possible de la chaîne d'analyse de la figure 1.5. Une architecture de type *Hadoop* permet tout d'abord de collecter et de filtrer des données à grande échelle sur du matériel standard par un paradigme *Map-Reduce* (voir chapitre 5). Les données retenues et pré-traitées ont encore une taille importante, mais sont supposée stockables sur quelques disques locaux. Elle sont alors déversées dans la partie *HPC*, qui permet d'entraîner efficacement un algorithme de *Deep Learning* sur des machines équipées de GPU.

1.4 Objectifs du cours

Ce cours de deuxième année de CentraleSupélec n'aborde pas tous les aspects introduits dans les sections précédentes. Il reste un cours d'informatique, et ne détaille pas les mathématiques statistiques utilisées en analyse de données. Mais, il aborde plusieurs facettes de l'informatique des *Big Data* :

- Des rappels sur les composants matériels des machines d'aujourd'hui (notion de cœurs, de processeurs, de clusters, de cloud...), et sur les composants logiciels (notions de tâches, de threads, de processus, et de synchronisation). La problématique du déploiement d'une architecture logicielle distribuée est introduite juste après ces rappels matériels et logiciels, avec notamment des objectifs de performance et de tolérance aux pannes.

- La définition de métriques de performances, et particulièrement de métriques orientées vers le *passage à l'échelle*. Certains modèles mathématiques de performances d'applications distribuées sont présentés.

- Les approches d'algorithmique distribuée et les outils les plus utilisés. Certains aspects concernent le filtrage et le pré-traitement des données à large échelle avec le paradigme de programmation *Map-Reduce*, ainsi que le fonctionnement interne d'*Hadoop*. D'autres concernent l'analyse de données à l'aide de l'environnement plus rapide *Spark* et de ses diverses interfaces de programmation. Enfin quelques principes algorithmiques et génériques de parallélisme, venant du calcul à haute performance, sont introduits rapidement, et se retrouveront dans des algorithmes simples de *Machine Learning* ou de *Data Analytics* étudiés en fin de cours.

- Un exemple d'architecture *Big Data* dans un environnement de *Cloud* est présenté avec la solution d'*AWS*, qui s'appuie sur l'architecture de stockage *S3*.

- L'évolution des BdD vers les bases *NoSQL* est détaillé, ainsi que les différents types de BdD *NoSQL* (orientées paires clé-valeur, ou documents, ou colonnes...). Plusieurs technologies de BdD *NoSQL* sont étudiées, ainsi que l'apport du paradigme *Map-Reduce* pour l'exploitation de ces BdD. La solution *MongoDB* est approfondie sur une machine et sur un cluster de PC.

– Une présentation des principaux algorithmes de *Machine Learning* termine le cours, avec une étude des facilités et difficultés de parallélisation de ces algorithmes dans un but de performance.

L'une des difficultés du *Big Data* est de traiter rapidement de très gros volumes de données. Les préoccupations de performance et d'optimisation des algorithmes et des codes jalonnent donc la conception et l'exploitation d'architectures logicielles de *Big Data*, bien qu'elles soient différentes de l'approche du HPC. D'une manière générale, les différences et ressemblances entre les approches *Big Data* et *HPC* sont signalées tout au long de ce cours.

Bibliographie

[1] G. Amdahl. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[2] B. Azarmi. *Scalable Big Data Architecture*. Apress, 2016.

[3] R. Bruchez. *Les bases de données NoSQL et le Big Data*. Eyrolles, 2ème edition, 2016.

[4] R. Bruchez and M. Lutz. *Data science : fondamentaux et études de cas*. Eyrolles, 2015.

[5] B. Chapman, G. Jost, R. Van Der Pas, and D. Kuck. *Using OpenMP*. The MIT Press, 2008.

[6] K. Chodorow. *MongoDB, the Definitive Guide*. O'Reilly, 2ème edition, 2013.

[7] K. Dowd and Ch. Severance. *High Performance Computing*. O'Reilly, 2nd edition, 2008.

[8] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. The MIT Press, 1999.

[9] J.L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31 :532–533, 1988.

[10] H. Karau, A. Konwinski, P.Wendell, and M.Zaharia. *Learning Spark*. O'Reilly, 1st edition, 2015.

[11] H. Karau and R. Warren. *High Performance Spark*. O'Reilly, 1st edition, 2017.

[12] M. Kirk. *Thoughtful Machine Learning with Python*. O'Reilly, 2017.

[13] P. Lemberger, M. Batty, M. Morel, and J-L. Raffaelli. *Big Data et Machine Learning*. Dunod, 2015.

[14] D. Miner and A. Shook. *MapReduce Design Patterns*. O'Reilly, 2013.

[15] T. White. *Hadoop. The definitive Guide*. O'Reilly, 3rd edition, 2013.

[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets : A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, 2012.