Méthodes d'apprentissage automatique appliquées à l'analyse des signaux d'utilisations des grands calculateurs

Théo Saillant

université
PARIS-SACLAY

Statistical and learning methods for the analysis of signals from HPC computer

Méthodes d'apprentissage automatique appliquées à l'analyse des signaux d'utilisations des grands calculateurs

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 574 : mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques appliquées
Graduate School : Mathématiques, Référent : ENS Paris-Saclay

Thèse préparée dans l'unité de recherche Centre Borelli (Université Paris-Saclay, CNRS, ENS Paris-Saclay), sous la direction de Nicolas VAYATIS, professeur des universités à l'ENS Paris-Saclay, la co-direction de Mathilde MOUGEOT, professeure des universités à l'ENSIIE, et le co-encadrement de Jean-Christophe WEILL, ingénieur-chercheur au CEA

Thèse soutenue à Paris-Saclay, le 24 novembre 2022, par

Théo SAILLANT

Composition du jury
Membres du jury avec voix délibérative

Laurent OUDRE Professeur des universités, Centre Borelli	Président
Madalina OLTÉANU Professeure des universités, CEREMADE	Rapporteur & Examinatrice
Angelo STEFFENEL Professeur des université, Université de Reims	Rapporteur & Examinateur
Raymond NAMYST Professeur des universités, Université de Bordeaux	Examinateur

THESE DE DOCTORAT
NNT : 2022UPASM029

université PARIS-SACLAY | ÉCOLE DOCTORALE de mathématiques Hadamard (EDMH)

Titre : Méthodes d'apprentissage automatique appliquées à l'analyse des signaux d'utilisations des grands calculateurs

Mots clés : Séries temporelles, Apprentissage, Statistique, HPC, Optimisation convexe

Résumé : L'objectif de cette thèse est de déterminer quelles méthodes statistiques peuvent actuellement être utilisées pour améliorer la compréhension de l'utilisation qui est faite d'un grand calculateur.

Nous décomposons le calculateur en trois parties : matériel, logiciels et utilisateurs afin de dégager trois pistes de recherches qui nous paraissait pertinentes.

Nous proposons un modèle permettant la prédiction de la consommation électrique d'un calcul avant qu'il ne soit placé dans la file d'attente, ainsi le logiciel qui gère cette file d'attente peut piloter la consommation du calculateur.

Nous cherchons également à visualiser plus facilement les données relatives aux évènements dans le calculateur qui peuvent être textuelles ou un nombre d'occurrences.

Enfin nous proposons de regrouper et découper des séries temporelles issues de senseurs posés sur les calculateurs du CEA.

Ces méthodes sont donc bien utiles pour les informaticiens et peuvent être originales pour les statisticiens.

Title : Statistical and learning methods for the analysis of signals from HPC computer

Keywords : Time series, Machine-Learning, Statistics, HPC, Convex Optimization

Abstract : The aim of this thesis is to determine what statistical methods can currently be used to improve the understanding of the use of a computing center.

We decompose the computer into three parts : hardware, software and users in order to identify three relevant research directions.

We propose a model allowing the prediction of the power consumption of a computer before it is placed in the queue, so that the software that manages this queue can control the computer's consumption.

We also seek to visualise more easily the data relating to events in the computing center which can be textual or a number of occurrences.

Finally, we propose to group and slice in relevent parts time series from sensors installed on the CEA's computers.

These methods are therefore very useful for computer scientists and can be original for statisticians.

Résumé en français

Le CEA opère des centres de calculs qui sont extrêmement sollicités en interne et par des partenaires scientifiques et industriels. Un enjeu important est la surveillance du bon fonctionnement des calculateurs HPC et de leurs périphériques. Pour cela le CEA a déployé une plateforme matérielle et une chaîne de traitement logicielle qui enregistrent et traitent de nombreux signaux temporels ou agrégés issus de ces matériels.

L'objectif de cette thèse à cheval entre les mathématiques et l'informatique est l'utilisation et la définition de nouvelles méthodes statistiques et d'apprentissage pour exploiter toutes les données d'utilisation des grands calculateurs afin mieux comprendre son utilisation : classifier les utilisations de différentes ressources, détecter les comportements aux limites, détecter les dérives ou pics d'utilisation.

Nous commençons par identifier les cas d'usages pour lesquels des méthodes statistiques auraient un apport significatif dans la gestion des grands calculateurs. Pour cela, nous décomposons le calculateur en trois grandes parties : le matériel, les logiciels et les différentes personnes qui interagissent dans le centre de calcul. Cette décomposition nous permet de contextualiser les données que l'on peut collecter et comment les exploiter. Cela nous permet de définir des pistes de recherche qui sont explorés dans le reste de la thèse.

Une première application est la prédiction de la consommation électrique d'une allocation de ressources avant même son ordonnancement grâce aux informations données par l'utilisateur dans sa demande de ressources de calcul au logiciel chargé de les ordonner. Nous détaillons pourquoi cette prédiction est indispensable pour piloter la consommation électrique du calculateur. Après avoir bien délimité quelles informations peuvent être utilisées et quelle quantité peut être prédite, nous proposons un modèle basé sur les instances passées pour réaliser cette prédiction. La simplicité de ce modèle nous permet de proposer une version particulièrement adaptée pour une utilisation en production. Enfin, nous évaluons la prédiction de la consommation globale future du calculateur de ce modèle une fois les allocations ordonnées.

Les experts utilisent souvent des données collectées sur les évènements telles que les journaux système ou les résultats de logiciels profiler pour mieux comprendre l'activité a posteriori. Nous proposons donc de faciliter l'analyse de ces données grâce à une analyse statistique et une visualisation. Pour cela, nous détaillons quels pré-traitements des données semblent nécessaires avant de pouvoir conduire une telle analyse et s'ils semblent envisageable avec le format

actuel des données collectées. Dans le cas favorable, nous proposons une transformation des données qui semble pertinente avant l'utilisation de méthodes d'analyse statistique et visualisation classiques. Nous décrivons alors les différentes possibilités ainsi que les compromis de chacune pour justifier le choix d'utiliser la projection du résultat de notre transformation sur les composantes principales comme visualisation des données liées aux évènements.

Enfin nous proposons un modèle de regroupement et segmentation de certains signaux temporels extraits par le CEA. Nous proposons d'abord une exploration et un modèle des ces données pour identifier les caractéristiques qui semblent pertinentes à extraire. Nous formalisons l'ajustement de ce modèle aux données comme un problème d'optimisation convexe et proposons une méthode de résolution spécifique plus rapide sur des grands signaux. Cette méthode de résolution nous permet d'utiliser un autre a priori qui semble plus adapté à nos données et à la segmentation.

Cette thèse conclut que les méthodes statistiques sont utiles pour la gestion d'un grand centre de calcul quand les données extraites sont appropriées à leur usage. Réciproquement, ce type de données peut motiver des modèles originaux dans le domaine de l'apprentissage automatique.

Contexte

Le progrès est ce qui fait l'histoire et l'humanité d'une société. C'est une évolution dirigée vers un idéal. Cette évolution se décompose en l'alternance de deux grandes taches : la mobilisation de la créativité humaine pour imaginer une situation désirée et sa confrontation avec la réalité complexe. Il est clair que l'expérience du monde physique est indispensable au progrès, mais la création d'un scénario potentiel est tout aussi nécessaire pour qu'une expérience fasse progresser la société ou qu'elle valide les anticipations du scénario. La modélisation est initiée par l'imagination d'un modèle. La manipulation du modèle se décompose en plusieurs opérations sur les quantités du modèle, appelées aussi "calcul". Mais ces opérations n'ont plus besoin de cette étincelle initiale d'imagination, bien que le calcul n'ait aucun sens en dehors du modèle imaginé par l'Homme. Ainsi, nous avons pu rapidement utiliser des outils, de simples petits cailloux comme le suggère l'étymologie latine du mot "calcul" par exemple, pour nous aider à manipuler ces quantités irréelles qui sont le fruit de l'esprit humain. Le calcul a toujours été le moteur du progrès, un moteur qui consomme la créativité humaine pour produire un scénario, une anticipation. Aujourd'hui, ce moteur est matérialisé par un outil présent dans la vie de tous, même dans les pays les plus reculés : l'ordinateur.

Avant de devenir un assistant pour de nombreuses autres tâches plus ou moins quotidiennes comme aujourd'hui, la fonction des premiers ordinateurs était de construire des représentations d'un modèle facile à confronter à la réalité. Le modèle était conçu pour être aussi précis que possible, c'est pourquoi ce processus est appelé "simulation". Ces représentations sont donc aussi complexes que la réalité pour être confrontées à elle, il est parfois impossible pour l'homme de les calculer par lui-même car le nombre d'opérations est trop important et l'homme, même le plus expérimenté, trop lent pour mener à bien le calcul. Ainsi, la puissance de calcul, c'est-à-dire le nombre d'opérations effectuées par seconde, d'un ordinateur est un facteur limitant du progrès, une ressource que l'on doit produire au maximum et dont l'utilisation doit être optimisée si l'on veut progresser plus vite, notamment dans le domaine technologique. C'est pourquoi, aujourd'hui encore, la course à la puissance de calcul est un enjeu majeur pour nos sociétés.

Dans le monde moderne, la simulation a surtout une application industrielle. Elle permet à l'industrie de réaliser l'ensemble de la conception d'un produit complexe dans le monde immatériel du numérique, réduisant ainsi considérablement le coût du prototypage, les conséquences d'un échec de conception et accélère la recherche et le développement de plusieurs ordres de grandeur en modélisant des situations qui ne pourraient jamais être expérimentées ou très rarement. Pour répondre à ce besoin croissant de l'industrie, des machines spécialisées pour fournir la plus grande puissance de calcul possible sont regroupées dans de grands centres de calcul et sont partagées entre plusieurs utilisateurs. Au fil du temps, ces superordinateurs sont devenus plus grands et plus complexes afin de produire de plus en plus de calculs à un rythme plus rapide, c'est pourquoi on les appelle les centre de calcul haute performance (high performance computing abrégé HPC en anglais).

L'administration des centres de calcul est donc une industrie à part entière avec une véritable production : le résultat des calculs, et ses usines remplies de moteurs du progrès technique : les ordinateurs. Contrairement aux ordinateurs à usage personnel, les acteurs humains qui gèrent le centre de calcul ne sont pas les mêmes que les utilisateurs de ces ordinateurs. La gestion du centre de calcul que les clients paient comprend le renouvellement des machines, la fourniture d'énergie pour leur fonctionnement, la gestion des externalités qu'elles provoquent... Mais la particularité de cette industrie est que ceux qui interagissent directement avec l'outil de production qu'est l'ordinateur le font souvent pour le compte des clients. Les utilisateurs imposent eux-mêmes à la machine une liste d'opérations à effectuer dans un format plus ou moins restrictif. Ainsi, contrairement à toutes les autres industries, la production est faite par les clients via les moyens de production et non par les employés du propriétaire de ceux-ci.

Bien entendu, l'administration de l'ordinateur contraint l'utilisateur afin qu'il ne puisse pas abuser de son droit d'utiliser la machine. L'administration peut aussi décider d'utiliser la machine avec des privilèges pour son propre bénéfice ou pour faciliter l'utilisation de la machine. Mais ce qui est payé par le client est le droit de l'utiliser pour effectuer les opérations autorisées qu'il souhaite, donc la rentabilité d'une telle industrie n'est possible que si les utilisateurs sont libres de demander à la machine d'effectuer leur séquence d'opérations. En d'autres termes, l'administration peut encadrer mais pas imposer strictement la manière dont les moyens de production seront utilisés pour produire.

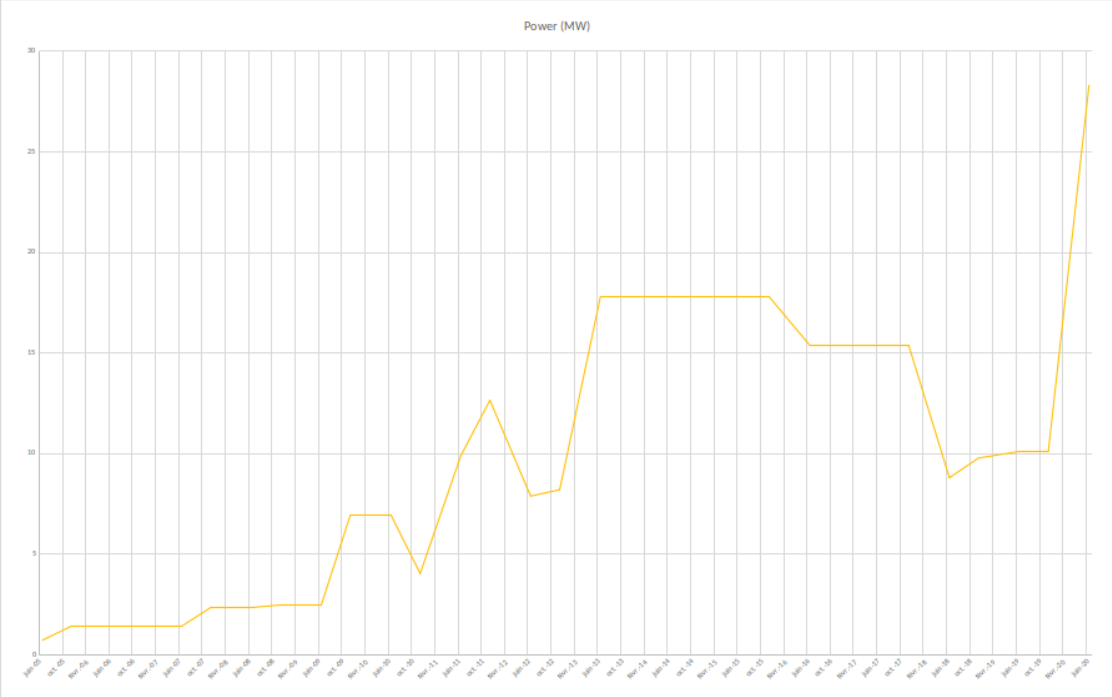
Cette séparation entre les utilisateurs et l'administration de la machine rend difficile la compréhension de son activité par l'administration. L'administration programme souvent les ordinateurs pour qu'ils enregistrent les événements liés à leur utilisation. Cependant, il n'est pas possible d'interpréter les opérations élémentaires tant elles sont nombreuses. De plus, les utilisateurs ne partagent généralement pas le détail des opérations qu'ils effectuent car ils sont des acteurs industriels et les codes et résultats qu'ils ont produits ont une valeur qu'ils ne souhaitent pas partager. Les opérateurs du centre de calcul ne peuvent donc ni savoir ce que les utilisateurs calculaient exactement au moment où les données ont été produites, ni savoir ce que fait exactement l'ordinateur au moment où il est en production sans autre information de la part de l'utilisateur. Les données produites par le suivi des machines sont donc nombreuses, mais mal structurées et souvent incomplètes. A l'heure actuelle, ces données sont principalement lues et utilisées par des opérateurs humains lors d'un incident pour caractériser l'origine du problème (on parle alors d'analyse forensique) ou pour concevoir le prochain modèle de supercalculateur. Ce travail est d'autant plus long et fastidieux que le supercalculateur devient de plus en plus complexe, ainsi que les codes qui produisent les événements enregistrés.

Cette impossibilité pour l'administration de déterminer le type d'utilisation de la machine par les utilisateurs n'a pas été un problème depuis le début de l'utilisation des premiers supercalculateurs et même jusqu'à très récemment. En effet, l'augmentation de la puissance de calcul a été principalement dominée par le développement de nouvelles machines plus puissantes. L'illustration la plus connue est bien sûr la loi de Moore, qui postule une augmentation exponentielle du nombre de transistors sur la carte mère. Pendant plus de 30 ans, la fréquence d'horloge des unités de calcul, c'est-à-dire la fréquence temporelle des opérations arithmétiques élémentaires à effectuer, a également doublé presque tous les 18 mois. Ainsi, la compétitivité du gestionnaire de l'ordinateur était beaucoup plus déterminée par sa capacité à changer régulièrement et rapidement les machines pour des modèles plus récents que par l'optimisation de l'administration de la machine. De plus, les programmeurs étaient beaucoup plus incités à op-

timiser leurs programmes en raison des contraintes plus fortes sur les autres ressources informatiques. Cependant, cette analogue de la loi de Moore n'est plus valable pour les fréquences d'horloge des processeurs, qui n'ont pas augmenté depuis au moins 2005. Seul le nombre de transistors sur une carte mère de même taille continue de croître de manière exponentielle et de faire face à ses propres défis. Cela signifie qu'un programme ne peut espérer tirer parti des récentes avancées matérielles que s'il peut distribuer efficacement le calcul à plusieurs unités en parallèle. Les programmes s'exécutant sur les ordinateurs HPC sont donc encore plus complexes et il est donc encore plus difficile pour l'administration de les caractériser sans connaître le code qui les a générés.

Mais la situation est en fait pire aujourd'hui et nécessite un changement. L'amélioration des équipements ne suffit plus car un nouveau facteur limitant est apparu récemment et concerne également les équipements récents : la consommation électrique. La puissance consommée par opération ne diminue plus de manière exponentielle comme auparavant. Maintenir une croissance exponentielle de la puissance de calcul nécessite aujourd'hui une croissance exponentielle de la consommation électrique comme illustré par la figure ci-après. Cette croissance n'est pas durable, un très gros ordinateur peut déjà consommer autant d'énergie que la production d'une centrale thermique. Au-delà de la croissance exponentielle du prix de l'heure de calcul en raison du coût de cette énergie, il est physiquement impossible d'augmenter la puissance de calcul dans un futur proche. Il est donc nécessaire d'adapter l'utilisation de l'ordinateur à l'énergie disponible et d'optimiser au maximum son fonctionnement. Il est donc nécessaire de réduire au maximum les indisponibilités liées à la maintenance ou aux incidents et d'optimiser l'allocation des ressources informatiques afin de réduire les pics de consommation voire de s'adapter à la production d'énergie disponible. Tout cela nécessite une bonne compréhension de l'activité de la machine : il doit être possible de trouver des synergies entre les différentes demandes des utilisateurs.

Mais l'interprétation des données dont nous disposons pour ce faire est trop longue et coûteuse pour être systématique car seule l'expertise humaine est capable de faire cette analyse. Dans le même temps, nous assistons à l'essor des méthodes d'apprentissage automatique et des techniques statistiques qui aspirent à mieux comprendre et contrôler des systèmes complexes produisant de grandes quantités de données. Ironiquement, ces techniques sont rendues possibles par la puissance de calcul disponible pour entraîner des modèles complexes et par des ensembles de données étiquetées massives. La question est donc de savoir s'il est possible d'automatiser totalement ou partiellement cette analyse avec des méthodes statistiques ou du moins de fournir des outils fiables pour faciliter la compréhension de l'activité de la machine tout en respectant



Évolution de la puissance électrique consommée par le calculateur le plus haut classé du TOP500 à l'instant considéré en MégaWatt (MW, échelle de 0 à 30 MW) entre Juin 2005 et Juin 2020.

les contraintes de production telles que la confidentialité des utilisateurs et les performances. Cela nécessite d'identifier les données qui peuvent être systématiquement extraites de la machine tout en respectant les droits des utilisateurs et en ne dégradant pas les performances de la machine.

Contributions

Les contributions de cette thèse à chaque chapitre sont les suivantes :

Chapitre 1 : Nous présentons une représentation simplifiée des différentes parties qui interagissent dans un centre HPC, utile pour comprendre les données qui peuvent être collectées et les applications possibles de celles-ci. Nous décrivons comment la politique tarifaire d'un centre de calcul peut être utilisée pour piloter la partie humaine du centre de calcul par le biais d'incitations. Nous proposons trois applications ou pistes de recherche de l'apprentissage automatique pour la surveillance des centres de calcul qui méritent d'être creusées.

Chapitre 2 : Nous décrivons quelles données disponibles pour l'ordonnancement peuvent être utilisées pour effectuer une prédiction de la consommation électrique de l'allocation d'un job. Nous avons proposé un modèle basé sur les instances ou mémorisation pour prédire la consommation électrique moyenne par noeuds d'un job, nous en avons fait une version d'estimation au fil de l'eau pour le rendre adapté au suivi de la consommation électrique de tout un centre de calcul dans un contexte de production et nous avons évalué ses performances. Ce travail a été publié dans l'édition 2020 de la conférence ISC-HPC.

Chapitre 3 : Nous constatons que l'analyse des journaux non formatés n'est pas un problème bien défini qui ne peut être résolu sans expertise. Nous avons proposé une méthode de visualisation des données d'événements hautement agrégées.

Chapitre 4 : Nous trouvons les caractéristiques des séries temporelles de communication brutes d'octets et identifions que la plus intéressante peut être extraite par un regroupement de sous-espaces après une normalisation bien choisie. Nous proposons une résolution efficace pour un modèle générique de regroupement de sous-espaces pour les séries temporelles et décrivons la version la plus appropriée de ce modèle pour nos données. Nous proposons une heuristique pour définir les hyperparamètres de ce dernier modèle.

Introduction

The CEA (Commisariat à l'Énergie Atomique et aux Énergies Alternatives) is in charge of three High Performance Computing (HPC) centers and has installed the necessary infrastructure to systematically collect and store data on the use of their computing resources using sensors and aggregation software. This data has already been used to optimize the specifications for the different computing resources of the next supercomputer models. However, this data has not yet been used to classify the different programs executed on the machine.

An interesting question is to better understand how the computing center is used and how this knowledge could improve the monitoring of HPC centers in general. To answer this question, we propose a concrete review of the potential contributions of statistical signal analysis methods on the set of data that can be connected and exploited in a production context.

We identify several applications that could have strong positive impact on HPC center management if the potential information contained in the data collected can be extracted. Our first work presents a review of the different parts and actors of a HPC center and what data is collected. From this review, we propose three research tracks on the computing center and the data available that are worth digging.

- The first track focus on the monitoring of the power consumption of the computing center. In order to predict the electrical power consumption of a user's allocation of computing node, we propose an instance-based model using the data the user provides when he submits his request. We also adapt the model to make it easy to update in industrial production context. The prediction of our model can be then used to reliably estimate the global power consumption of the computing center when combined with the schedule of allocations. This implies our model could be used to schedule resource allocation while keeping the power consumption of the computing center under control.
- Our second research track is the visualization of log and event data. We first show that the automatic analysis of log data may not be reliable without any human preprocessing or supplementary information on the log generation. However, we show that when reliable error codes or identifiers are available to identify which events are the same, the data can be presented in a very simple but helpful way for the human analyzing such data. But events data with such an aggregation seems to have an original behavior not well studied in machine learning. We describe them as high

count data and we propose several consistent embedding of such kind of data to better visualize it. Further research to mine such data would require labeled data.

- The last research track is an unsupervised model of the time series data collected by sensors. We first choose to focus only on the number of incoming bytes to a computing node and review what interesting features may be extracted from the data. We propose a model to cluster nodes implicated in the same allocation and segmented their time-series so that more reliable estimators of computing resources utilization can be build. We accelerate the resolution of the model using convex optimization techniques. They open up the path to variant to better match the prior knowledge we have on the data which ends up being interesting problem for the machine learning field.

All this work is the first step toward practical applications of machine learning for HPC management.

Context

Progress is what makes the history and the humanity of a society. It is an evolution directed towards an ideal. This evolution is decomposed in two great stages : the mobilization of the human creativity to imagine a desired situation and its confrontation with the complex reality. It is clear that the experience of the physical world is indispensable to progress, but the creation of the scenario is just as necessary for an experience to make society progress, whether it validates the anticipations of the scenario. The modeling is initiated with the imagination of a model. The manipulation of the model is broken down into several operations on the model's quantities, also called "calculation". But these operations no longer need this initial spark of imagination, although the calculation has no meaning outside the model imagined by Man. Thus, we were able to quickly use tools, simple little stones as suggested by the Latin etymology of the word "calculus" for example, to help us manipulate these unreal quantities that are the fruit of the human mind. Calculation has always been the engine of progress, an engine that consumes human creativity to produce a scenario, an anticipation. Today, this engine is physically embodied by a tool present in everyone's life, even in the most remote countries : the computer

Before becoming an assistant to many other more or less daily tasks like today, the function of the first computers was to build representations of a model easy to confront with reality. The model was designed to be as accurate as possible, which is why this process is called a "simulation". These representations

are therefore as complex as reality to be confronted with it, it is sometimes impossible for man to calculate them by himself because the number of operations is too great and man, even the most experienced, too slow to complete the calculation. Thus, the computing power, the number of operations performed per second, of a computer is a limiting factor of progress, a resource that we must produce to the maximum and whose use must be optimized if we wish to progress faster, in particular in technology. This is why even today the race for computing power is a major challenge for our societies.

In the modern world, simulation has mainly an industrial application. It allows the industry to realize the whole design of a complex product in the immaterial world of digital, reducing considerably the cost of prototyping, the consequences of a design failure and accelerates research and development by several orders of magnitude by modeling situations that could never be experienced or very rarely. To meet this growing need in industry, specialized machines to deliver as much computing power as possible are grouped together in large computing centers and are shared among several users. Over time, these supercomputers have become larger and more complex in order to produce more and more calculations at a faster pace, which is why they are called high performance computing (HPC) centers.

The administration of computing centers is thus a full-fledged industry with a real production : computing, and its factories filled with engines of technical progress : the computers. Unlike computers for personal use, the human actors who manage the computing center are not the same as the users of these computers. The management of the computing center that the customers pay for includes the renewal of the machines, the energy supply for their operation, the management of the externalities they cause... But the uniqueness of this industry is that those who interact directly with the production tool that is the computer often do so on behalf of the customers. The users themselves impose a list of operations to be performed in a more or less restrictive format on the machine. Thus, contrary to all other industries, the production is made by the customers via the means of production and not by the employees of the owner of these. Of course, the administration of the computer constrains the user so that he cannot abuse his right to use the machine. The administration can also decide to use the machine with privileges for its own benefit or to facilitate the use of the machine. But what is paid for by the customer is the right to use it to do the permitted operations he wants, so the profitability of such an industry is only possible if the users are free to ask the machine to perform their sequence of operations. In other words, the administration can frame but not strictly impose how the means of production will be used to produce.

This separation between the users and the administration of the machine

makes it difficult for the administration to understand its activity. The administration often programs the computers so that they record the events related to their use. However, it is not possible to interpret the elementary operations as they are so numerous. Moreover, the users do not generally share the details of the operations they are performing because they are industrial actors and the codes and results they have produced have a value they do not wish to share. The operators of the computing center can therefore neither know what exactly the users were calculating at the time the data was produced, nor know what exactly the computer is doing at the moment when it is in production without further information from the user. The data produced by machine monitoring is therefore numerous, but poorly structured and often incomplete. At present, this data is mainly read and used by human operators during an incident to characterize the origin of the problem (this is called forensic analysis) or to design the next supercomputer model. This work is all the more time-consuming and tedious as the supercomputer becomes more and more complex, as well as the codes that produce the recorded events.

This impossibility for the administration to determine the type of use of the machine by the users has not been a problem since the beginning of the use of the first supercomputers and even until quite recently. Indeed, the increase in computing power was mainly dominated by the development of new, more powerful machines. The most known illustration is of course the Moore's law, which postulated exponential increase of the number of transistors on motherboard. For more than 30 years, the clock frequency of the calculation units, i.e. the time frequency of the elementary arithmetic operations to be performed, also doubled almost every 18 months. Thus, the competitiveness of the manager of the computer was much more determined by his capacity to change the machines regularly for more recent models quickly than by the optimization of the administration of the machine. Moreover, the programmers were much more incited to optimize their programs because of the stronger constraints on the computing resources. However, this kind of Moore's law is no longer valid for clock frequencies, which have not increased since at least 2005. Only the number of transistors on a motherboard of the same size continues to grow exponentially and face its own challenges. This means that a program can only hope to take advantage of recent hardware advances if it can efficiently distribute the computation to several units in parallel. Programs running on HPC computers are therefore even more complex and it is therefore even more difficult for the administration to characterize them without knowledge of the code that generated them.

But the situation is actually worse today and requires a change. The improvement of the equipment is no longer enough because a new limiting factor has re-

cently appeared and also concerns the recent equipment : the power consumption. The power consumed per operation is no longer decreasing exponentially as it used to. Maintaining an exponential growth of the computing power requires today an exponential growth of the electrical consumption. This growth is not sustainable, a very large computer can already consume as much energy as the production of a thermal power plant. Beyond the exponential growth of the price per hour of computing because of the cost of this energy, it is physically impossible to increase the computing power in the near future. It is therefore necessary to adapt the use of the computer to the available energy and to optimize its operation as much as possible. It is therefore necessary to reduce as much as possible the unavailability linked to maintenance or incidents and to optimize the allocation of computing resources in order to reduce consumption peaks or even to adapt to the available energy production. All this requires a good understanding of the machine's activity : it should be possible to find synergies between the various user requests.

But the interpretation of the data available to us to do this is too long and costly to be systematic because only human expertise is able to make this analysis. At the same time, we are witnessing the rise of machine learning methods and statistical techniques that aspire to better understand and control complex systems producing large amounts of data. Ironically, these techniques are made possible by the computing power available to train complex models and by massive labeled data sets. The question is therefore whether it is possible to fully or partially automate such analysis with statistical methods or at least provide reliable tools to facilitate the understanding of the machine's activity while respecting production constraints such as user confidentiality and performance. This requires to identify the data that can be systematically extracted from the machine while respecting the users' rights and not degrading the machine's performance.

Main contributions

The contributions of this thesis by chapter are the following

Chapter 1 : We present a simplified representation of the different part interacting in a HPC center useful to understand the data that can be collected and the possible applications of it. We describe how the pricing policy of a computing center can be used to monitor the human part of the computing center through incentives. We propose three applications or research tracks of machine learning for the monitoring of computing centers worth digging.

Chapter 2 : We describe which data available for scheduling can be used to perform a prediction of the electrical power consumption of a job allocation. We proposed an instance based model to predict the average power consumption per nodes of a job, made an online version of it to make it suitable for the power consumption monitoring of a whole computing center in a production context and evaluate its performance. This work has been published in the 2020 edition of the ISC-HPC conference.

Chapter 3 : We find that the parsing of unformatted logs is not well-defined problem that cannot be solved without expertise. We proposed a method of visualization of highly aggregated event data.

Chapter 4 : We find the features of raw byte communication time series and identify that the most interesting one can be extracted by a subspace clustering after a well-chosen normalization. We propose an efficient solver for a generic model for time series subspace clustering and describe the most appropriate version of this model for our data. We propose a heuristic to set the hyperparameters of this last model.

Table des matières

Table des figures

Liste des tableaux

1 - Issues and data for monitoring HPC centers

Our topic of interest is the monitoring of a particular type of computing facilities : the High Performance Computing Centers. This work may be applied on any kind of High Performance Computing centers, whether they are performing computation on floating numbers for simulation or on integers for cryptography. These centers aim to provide the most computing power possible to its users which implies constraint on available memory, disk read and write speed and interconnections between very efficient programmable chips or integrated circuits called processing units.

Other types of computing facilities may focus on data storage, like data centers for data replication and cloud storage which rather constraint storage size and disk read speed, or good connectivity, like web server hosting, cloud gaming or financial server which rather constraint network latency and the location of the facilities. We do not expect our work to apply well on such type of facilities because the needs are very different. We won't be referring to such facilities when using the name "computing centers".

In order to find where machine learning can help the operation of a computing center we need to formalize how the computing center can be described from a high level perspective. A computing center is an industrial system and any such system is complex. We need to formalize it to have a better understanding on how it behaves and to identify what can be improved for production.

We first decompose the computing center in main parts. Then we discuss what data can be observed and collected about the interaction between them. This leads us to propose several research tracks for which we identify conditions for machine learning to be relevant in monitoring.

1.1 . Formalization of the different parts of a computing center

A computing center can be decomposed in three parts interacting with each other : the machine itself, the software running on it and the human users of the machine.

The machine is the simplest part. Its behavior is given by the laws of physics which are independent of the software or human.

1.1.1 . Breaking up the hardware of a computing center

The machine itself is already a complex system. The main components are the computing nodes. We introduce what they are, how they are organized as a network and its two kinds of inputs and outputs : data and electricity.

The computing nodes

The industrial production of an HPC center can be seen as the result of the execution of a high number of elementary instructions required by a user. A **core** is a set of electrical circuits capable

of decoding and executing any sequence of instructions called a **program**. The core will be considered as the smallest unit of a computing center in this work, but it is often composed of several smallest computational unit specialized or not in the execution of some common instructions (like multiplication or vectorized computation...). The instructions are executed at a rate given by a clock (a physical resonating quartz) so the rate of execution of a sequence of instructions is called **clock frequency**.

Not all instructions have to be executed in one sequence, instructions are often split in several sequences so several cores can be used to execute a given set of instructions. We call **parallel computations** or **parallelism** the fact that some instructions of a given set are executed at the same time thanks to several cores. The core is the smallest computation unit of a computer which contains often 2, 4 or 8 core units in the case of Personal Computer. An HPC center maximizes its production by using many cores.

Cores by themselves are not capable of storing instructions and results of computations. The instructions and data are stored in **memory** which can be read or written. The core are packaged in integrated circuit already containing memory with very fast access rate in the form of registers and several caches (organized in a hierarchy from L1, L2 and sometimes to L3). Such integrated circuits are called **CPUs** (Central Processing Unit). CPUs are considered the most generic type of Processing Unit, they execute any program with reasonable performances. But cores and memory amount and distribution can be more or less specialized, often at the cost of being less generic, resulting in other type of chips which are called differently : Graphical Processing Unit (GPU) specialized in computational graphics at first, Associative Processing Unit (APU) for massively parallel data processing or Tensor Processing Unit (TPU). Some computer may be dedicated to execute one very small set of instructions very efficiently like Field Programmable Gate Array (FPGA) or even Application-Specific Integrated Circuit (ASIC) used for cryptocurrencies mining. There are also new prototypes like neuromorphic chips or Quantum Processing Unit where the concept of instruction can be quite different from other chips. The HPC centers considered in this work and in the general case mostly use CPUs and GPUs but all the considerations on the monitoring of HPC center can be made for any type of chip.

The memory inside CPUs are rarely used by the user to store the instructions he wants to execute and are more reserved for a better management of the cores. Other memory sockets called **RAM** (Random-Access Memory) are used to store and transmit the instructions to the CPU through a shared communication bus, a data highway. All of these circuits are put on a motherboard which also include components to deliver electrical power, cool down circuits if needed and communicate with other devices. The resulting device is a computer. In HPC a **computing node** is a computer as a member of a network which can be used to execute together a set of instructions.

The numbers of cores, clock frequency and amount of memory in caches or RAM must be chosen when building the node. These features will have implications on the rate at which a set of instructions will be executed. An HPC center can have different types of nodes for different types of applications. For example, genetics computations often required to store long sequences in memory that must be stored in RAM to be accessed, this means the size of RAM will be a limiting factor of a node while massively parallel applications are limited by the number of cores. A **partition** is a set of nodes which all have the same type in the HPC center.

Distributed computing architecture of nodes

The computing nodes of an HPC center must be able to communicate with each other to be able to run a program together. The nodes are connected to each other thanks to high-quality wired network called the **interconnection network**. The topology of this network is often designed to minimize the latency of a subnetwork of given size, to increase the overall bandwidth and to minimize the interaction between two independently allocated subsets of nodes (it is often the fat-tree, the butterfly or the tore). The topology of the interconnection network implies variation of latency between any pair of nodes depending of the number of switches on the shortest path between two nodes. In this work we won't consider this effect and use a simplification where the interconnection network is a simple intermediary between nodes which is agnostic to the pair of nodes considered.

The interconnection network adds another layer of parallelism on top of nodes but with more constraints. The parallelism inside a node can be based on multiple threads, where a **thread** is a sequence of instructions which can be executed independently. All of the threads executed by a node can be part of the same **process**, an instance of a program, and they will share the resources (core and memory) assigned to the process by the node. However a process cannot be shared between several nodes since there is no memory shared by all nodes directly accessible to the CPUs. This means that the parallelism layer between nodes can be used only by several processes and the resources are distributed according to the ones available on each node.

The different inputs and outputs of the computing network

The HPC center also features auxiliary services to interact with the computing nodes. The function of these services is to provide instructions and data to process by the computing network.

The most important auxiliary service is the **File System**, like NFS specified by [Shepler et al., 2003]. The file system is a set of devices which write data on permanent physical support. This allows the data to persist when system is shutdown and to transport the data. The file system is often a set of hard drives (HDD) or solid state devices (SSD) in the case of personal computer. In the case of an HPC center a combination of several types of storage are used depending on the trade-off between the access speed needed and the failure rate. An interesting illustration of this trade-off is the use of magnetic tapes to store results for a very long time because it is very reliable despite the slow access rate and low price.

The file system deserves a particular attention. It can be a limiting factor in computation speed because the access to data stored in file system is very slow compared to the cached and RAM memory but it must be used to store high amount of data between shutdown. The memory is also a significant source of failures which are purely random because they come from wear rates and it can be very hard to detect them in advance.

The other main class of auxiliary services are connection utilities. They are servers acting as gateway between the computing center and the public Internet from which users connect to interact with the nodes. Users can also use specialized nodes in graphical processing to interactively render the result of their simulation.

Other servers are used for administration purpose. In the case of the studied HPC center, a server is in charge of collecting utilization statistics and store them.

Power supply and distribution

The HPC center must be continuously supplied with electricity to work. The power used is provided by the electrical grid at industrial scale. The current is distributed to nodes, servers and storage systems to keep them up. The electrical supply is by far the main cost of the biggest HPC centers business model [Pospieszny, 2012]. The electrical consumption is naturally increasing with the amount of computation.

A big part of the power supply is dissipated by devices as heat. This heat must be driven out so that the nodes stay at the optimal temperature. Heat pipes often in copper are used to conduct the heat to heat exchangers. The heat exchangers are close to a cooling source which can be air or liquid. The cooling source is maintained cold by a big cooling system which is often a pump which transfer the heat to the naturally present water or cold air around the location site of the HPC center and reject it as waste.

The COBALT and IRENE computing centers

Our data comes from two very large High Performance Computing centers which are hosted by the CEA called IRENE and COBALT. The complete hardware specifications of the two computing centers can be found at [CEA, 2022b] and [CEA, 2022a] respectively.

Both computing centers are for research purpose, IRENE is focusing on public domain while COBALT is dedicated to the industrial research. More details on their hardware and other contemporary HPC centers can be found in [Vetter, 2013].

1.1.2 . Distinction of the different software

To be executed programs must be send to the computing resources. This is made easy for human thanks to other programs. We classify the functions fulfilled by software running in a computing center in three classes depending on how they interact with hardware and humans and who run them.

System software and resources management

Most of the system software are used to optimize the resources management and hide the complexity to the user behind simple interface. In particular, the users can use different type of file systems with very little practical difference thanks to file system management software like **HPSS** (High Performance Storage System) which is able to handle the specificities of each storage system instead of the user.

On a personal computer, resource management software are all system software and humans rarely interact with such software. However, for shared computers like an HPC center there must be a resource management software which assign resources and it must interact with users. Users do not have a direct access to computing nodes. Servers are used as a gateway to allow users to interact with the nodes. This allows to regulate access to the computing resources through administration software. Almost every computer regulate the access to its content with a system of privileges. The operators of the computing center have access to all the features while users are limited for cybersecurity reasons.

To connect to the HPC center, a user must have an account. But having an account is not sufficient to be able to run a program. When the user logs in to the HPC center, he ends up connected to a login node with generally insufficient resources for the program he want to run. To launch his program on potentially several computing nodes, he must submit a sequence of instructions to the **Resources and Jobs Management System** (RJMS). It is a text file of commands

called a **script** which instructs the programs to launch and in which order. The simplest kind of script is a command line, an executable with its arguments. The users also add criteria on the duration of the resource allocation, the number of nodes needed to run all the programs and the maximum number of processes allowed if relevant. The user can also ask to interact directly in real-time with the node instead of running a script. An allocation of resources by a user is called a **job**. Each call to an executable during the job is called a **step**.

The RJMS fulfills the function of a job scheduler. It collects all the job submissions and defines an order in which the resources will be allocated. The way the order is set depends on the policy implemented on the computing center. The scheduling policy may be defined by a priority associated to each user or group of user, it may also depend on the previous job allocations of the same user or group of users.

Shared software for node

Given the current complexity of the programs, no developer specifies entirely the sequence of instructions to execute. Instead, they use language compilers to write a source code that is understandable by human then translated in the machine language of the nodes. They also used **libraries** of functions which are trusted subsequences of instructions to realize and manage certain computations. Those programs are very often launched by many users and they are made available by the computing center administrators to users. They are known and trusted by the HPC center operators in general. Some users may use open source libraries (source code is available and free to use) or known programs from ISV (Independent Software Vendor).

The administrators of the computing center may also provide other software to facilitate optimal settings for the available hardware and certain tests. For example, users may not have direct access to the complete status of the machine but may use a profiler furnished by the computing center operators.

Application software

Once the RJMS allocated resources to a user, it will execute the programs in machine language the users asked him to execute in the submitted script or open an interactive session for the user. Because the sequence of instructions is coded in machine language, it is very hard to reliably identify what are the instructions exactly for the administrator. It is possible to extract name of functions also called symbols in the executable but users can choose unusual names and this can be considered as retro-engineering which poses legal concern. This acts as a natural limit of confidentiality for the user which generally does not want to share the details on what is computed.

Although the users programs are not editable, they often use the shared libraries described previously. In HPC, the users makes generally numerical simulation and run known industrial production code with the optimized libraries installed on the computing center. The libraries can be instrumented by the administrators to log some event that they can use to improve the computing center management or better understand the users need.

The users' software are probably where the administration of the computer center has the least information and control.

1.1.3 . Understanding the human actors of a computing center

An HPC center is more than just a lot of hardware running a lot of software : it is maybe the most representative instance of a business model where a resource (here the computing power

of the HPC center) is better used when shared. The pricing policy is the key element directly in the hand of the computing center administration which drives how the users will interact with the hardware through software. However, the users may also have incentives on which the administration has less control.

Economics of an HPC center

The rise of cloud computing in 2010 was already predicted [Harms and Yamartino, 2010] by the fact that it allows to get an updated information system with high performance and quality of service at low cost [Gupta et al., 2013] and less ecological footprint. One reason is that the product of a computing center is a highly user specific, intangible, non-rival and non-excludable asset : once the results of the execution of the exact program the user wanted to execute is out, there is no need to redo the computation ever again. This means that users of computer will do their computation only once in an ideal case and do not need the computing resources anymore. It implies that a computing center is more profitable when shared with a sufficient number of users to always fill the jobs schedule of ideal computations that users need to do once. This profitability will then benefit to every user since it allows concentrating efficiently more computing power and human expertise to maintain them in one place thanks to significant economies of scale, allowing new possibilities at a lower price for everyone. When production is information itself, everyone wins by sharing the means to produce the information.

The incentives of human actors are not external to an HPC center, they are a feature of the HPC center to align the users and administrators self-interest with a more efficient use of computing resources. These incentives are asymmetric gain and cost between users and administrators. The sharing of computing resources aims to transform users' CAPEX (capital expenditure) into OPEX (operational expenditure) when their need are not enough to justify that they build and maintain their own computing center. It implies that the main incentive is the pricing policy of the computing center. Different pricing policy implies different users with different behavior.

Pricing as a tool of administration

The different pricing of HPC centers usage can be mainly split into three categories :

— Pay-as-you-go or On-demand pricing
  This model is often used in cloud computing, the primary example is the main offer of Amazon Web Service [AWS, 2022]. The users are charged per unit of time of allocation of the computing resources. The main feature of such pricing policy is that there is no underutilization of computing resources from users point of view. It provides a lot of flexibility to users but it also incentives them to minimize their use of the computing resources. This type of allocation imposes more constraints to the administration point of view since it is hard to predict demand, and it may quickly end up in situation where users compete for the same computational resources at the same time.
  To smooth the spikes of demands, the service provider sells saving offers to users if they reserve instances in advance, use more than a certain amount of resources in a fixed time period or allow their computation to be temporarily stopped if needed (using a notification system). The usage of resource can then be anticipated in the short term to fill more easily the job schedule and incentive the users to run their job when the constraints are low from administration point of view.
— Monthly rent, Project assignation
  Cloud computing providers like OVHcloud [OVHcloud, 2022] can also propose to pay a

monthly rent for a user-dedicated server. The advantage is that the user does what he wants and there is no job scheduling to do by the administration. However, the production may not be maximal, the user may sometimes not use the computing resources he paid for.

The computing centers for research often assign a quota of computing hours to a research project which must be consumed over a given time period. This means the computing hours are paid no matter what.

In the case of public research, the quota is often given to users based on their project submission as in the Juliot-Curie computing center. For private and industrial research, the companies can pay a share of the computing center and they get the same share of the computing hours of the center. This is the case of COBALT computing center. In such a quota systems, there may be a penalty for users which do not use the computing resource to attract users with a real need of computing hours at a cheap price.

— Free of charge with priority

Some industries (applications in military, meteorology, geology prospection) know they will always need more computing power at any time. In that case, it can be interesting for the company or institution to have its own computing center, as Total and its computing center Pangea III. The users and administrators are then company employees. Multiple sharing system can be used for such computing center.

In particular, a priority queue can be set up by the administration to define which activity of the company get more computing power depending on their need and the need of the company.

Another case is when the requirements are very specific to the industry itself. The financial sector needs to have their servers as close as possible to the stock exchange to reduce latency as much as possible and may use dedicated servers.

These three categories of pricing policy imply very different incentives to users because they change the cost structure of a computation and we can expect that the usage of computing power will be very different if the pricing policy is different.

However, the cost structure is almost the same in every case from the administration point of view. The main cost is more than before the electrical energy supply which is approximately proportional to the number of computing hours. But the marginal cost can be higher if the electrical power supplied spikes at high values : the same amount of electricity is often more costly if it is delivered in spikes than when it is supplied uniformly in time. The administration must also perform maintenance regularly and fix any breakdown which reduce the availability of the computing center and can be interpreted as an additional loss. Finally, building the computing center is an initial cost that must be compensated when pricing the usage.

The pricing policy can be a tool to reduce the cost of administration of the computing center to increase overall profitability. It incentives the users to constraint their usage against a reduction of their cost. Knowing how the users will use the computing center allows the administration to reduce all the costs. The pricing policy can smooth power spikes, allows anticipating when the regular maintenance can be done to avoid rejecting users asks for computing power as much as possible and to reduce machine requirements when possible which reduces the building cost.

Sociology and Human Behavior

The pricing policy is a driving force of the users' behavior but it has its limit to help the administration of the computing center. The user is human and can not be described as a simple

agent maximizing a utility because the computing center is a shared means of production. What a user does, can impact what others users will be able to do.

The users can try to compensate a bad choice of pricing policy for them. If they overestimated their needs, they may have a penalty. Such users may run programs like cryptocurrency mining which can be run on any type of computing resources to avoid penalty and get a little financial gain from the burned computing resources. This behavior can be considered as malicious because the cost is carried by the administration of the computing center and the others users may have trouble to run legitimate programs. Other malicious programs like malware can run against the will of the user too. It is in the best interest of everyone that the administration detects and shutdowns such programs.

The user decides which type of computing resources he will use and which programs to run to produce the results he wants. But because of many practical reasons, he may not use the best hardware or software available to get the result. One reason is that changing the industrial production software is a very long process because the users must evaluate if the new code is really the best choice and learn how to use the new one if they decided that it was worth the change. Not using the best software impacts the user production but also the production of other users because a user using the suboptimal software will use the computing resources for himself longer than needed. This means that it is in the best interest of the administration of the computing center to propose working sessions to promote new hardware and software which are known to be more efficient for users applications.

A pricing policy can help to monitor the computing center but it is not sufficient at all. To be efficient, we need to include expert knowledge and information about what the users intend to do. A better understanding of the human actors using the computing center could make the monitoring more efficient and robust through a personalized pricing policy using data or meta-data collected on the user and its jobs.

1.2 . Contextualization of the collected data

To improve the monitoring of the computing center beyond the incentives of a simple pricing policy, precise data about computing center usage is needed. We can first use the meta-data about submitted jobs logged by the RJMS, which contains no information on the content of the jobs themselves but about the typical usage of the computing center by a user or a group of users. The administration of the computing center can also use sensors to collect data about the usage of the resources and use them to guess what the users need. Finally, data about the content of the jobs themselves are hard to obtain because the users rarely share the source code of their software but the codes can be slightly instrumented by changing the shared libraries used to return interesting aggregated measures in what is called system logs files which are already used for maintenance.

1.2.1 . Usage pattern from Job scheduling data

The RJMS is an administration software so it is easy to collect meaningful data on how it makes its task. However, data on the job content seems to be irrelevant in practice. Only the meta-data (data about the job but not its content) can be used as feature for a prediction of other data we can only collect once the job is over.

Constraint on collected data and malleability

When users want to run a job, they have to submit their request to the RJMS. The request often take the form of command line which call a text file with high-level sequence of instructions to run. A such text file of commands is called a script.

The script will instruct the RJMS which executable files of the user must be run once the resources are allocated by the RJMS. The executables are unknown before they are started and it is often hard to guess what an executable will do just from its name for various reasons :

— The users may use the default name for the output of a compilation (which is often `a.out`)
— The executable could be the interpreter of a language (like `python`). The relevant name is then the file used as input argument of the executable but we can't be sure which argument is relevant
— Users may use the name to differentiate executables which result from different compilation settings for his own convenience

That is why it is difficult to use the instructions inside the script for monitoring : the script itself is noisy and user-malleable. Another idea [Yamamoto et al., 2018] is to run simple decompilation tools like symbols extraction (the name of functions in the executable) to identify if two executables come from the same source code but the exploitation of such information is hard because it can be very noisy and the user has no incentives to use the same symbols.

This means that we must carefully focus our attention on meta-data related to job submissions to better understand the users' utilization of the computing center.

User submitted and Job consumption data collected

Meta-data are much more reliable because the users cannot change them without practical consequences on the computational performance. The users need to ask for a precise number of nodes and processes so that what they submitted run correctly. This data is directly related to what the user want to do, it can be trusted.

Other data provided by the user may or may not provide information on how he wants to use the computing nodes depending on the computing center. Such data is the hard timelimit of a job after which it is killed by the RJMS. The value is needed to schedule jobs. But on computing center like COBALT, users often let this value at default (which is the maximal one) because there is no strong incentive for users to give a right estimate. It often gives no information on the duration of the job but some computing center could incentive users to input a more representative value of the job duration by charging the computing hour as a ratio of the timelimit set by the users.

Finally, some data about the job itself are available only once it is over.

Aggregation with RJMS logging

Once a job is over, some observation can be easily recorded to have a better understanding of the usage. The RJMS can record the true job duration, which could not be used to schedule the jobs. The RJMS will often return the data aggregated at the whole job level as a log.

Those data are combined with meta-data and logs information. The RJMS will log the user ID and a timestamp about when it received the submission. In the case of COBALT, the user is member of a group and the ID of the group is also recorded. Then it combines these with data about the time on the submission itself and the aggregated data about the job. The resulting data is about 4000 jobs a month, each job has about 15 features.

The resulting data summaries how much the computing center was used in average at the given period of allocation. It can be used to better understand the usage of a computing center

by user. However, it would be difficult to use such data for other computing centers because the user and group ID are specific to the computing center. The type of computing center also implies different behavior in general, we may have less diversity of codes in industrial computing centers than in a public research computing center. This implies that the information we can extract about usage pattern is very different depending on what is known on the users of the computing center.

1.2.2 . Information extraction with Communication data

Job metadata is useful to understand the habits of users but not the content of what they submitted. While it is not possible to get the details on the instructions that a program contains, it is possible to monitor how much it consumes computational resources of a node which could be useful to understand users' utilization of nodes. One particularly interesting metric is the number of bytes exchanged between nodes already studied in HPC research field. However, it is not directly available, so we cannot apply the same analysis.

Collection of data at node level

With sensors on the nodes of the computing center it is possible to collect signals of usage of the computing resources of a node during a job allocation. In the case of COBALT computing center, more than 500 metrics are collected on 2000 nodes. Those metrics are directly related to the code that the user is running. One advantage of using such metrics is that a job classification model trained on the data of one computing center may be easier to use to classify job from the same kind of data of another computing center because users tend to use the same production code for their application no matter the computing center.

Some metrics collected on each node are CPU usage (in system time, user time and wait), allocated RAM, the number of certain functions' calls, power usage... For most of the possible metrics, the values are sampled or aggregated every 5 seconds. They are sent to a statistic cluster that runs a log aggregation software like Logstash [B.V., 2022] which aggregates the data in time series indexed by job to store them.

One very interesting metric to attempt to characterize jobs is the count of the bytes coming in and out of a node. It is collected by the network card of each node. The bytes count's values are regularly sent to the log collector and it computes how many bytes was sent and received by each node. As most of the measurements, the bytes counts are sent every 5 seconds. It is a trade-off between data precision and storage and limited bandwidth of the interconnection network. Given this data, as it is, it should be possible to compute features from it that enable an analysis of applications based on the data exchanged between nodes with tools of HPC.

Distinction with communication in HPC context

The importance of communication in HPC is the reason why the bytes count is an interesting metric.

In HPC, the **communications** refers to the data exchanged between workers which can be threads or processes. The communications between nodes can be a limiting factor in computing speed. Indeed, the transmission of data is slow compared to CPUs frequency and RAM memory access. This explains the expansion of a whole field of HPC research dedicated to the study and optimal reduction of communication loads in **applications**, the set of processes running together. Two widely used libraries in HPC, MPI [Gabriel et al., 2004] and OpenMP [Dagum and Menon, 1998], allow managing communications between threads and processes respectively. The

communication management of production ready application is generally very optimized and strongly dependent of the type of application.

The way distributed resources and workers are managed define the **Parallel Programming Model**. While communicating workers are necessarily processes when they are on different nodes, the workers inside nodes can be either processes or threads. We say the parallel programming model is **Full MPI** in the latter **hybrid OpenMP MPI** in the former. There are several paradigm to distribute tasks, the most known is **client server** distribution model (also known as master and slave model) with the **Peer-To-Peer** model.

The communication patterns of an application that can be observed are strongly related to the Parallel Programming Model and distribution model it uses. Those pattern has been studied using a specialized profiler like [Knüpfer et al., 2008] mainly as a tool to optimize a given application. The extraction of precise communications pattern often requires to instrument the source code and imply an overhead which limit the ability to let it run during production. It is possible to discriminate the different running applications with the reporting of such communication.

That's why we could expect that the evolution of the number of bytes received and send by each node may contain characteristic patterns of certain jobs since communications are aggregated in such metric.

Distinction between applications by the raw inter-nodes communication pattern

However, the data we are collecting is not the same as in the case of study on HPC communications.

The byte's counts are raw measure of information exchanged between nodes. There are several differences between our data and the communications in the sense of HPC :

- The network card is not aware of the data exchanges inside each node between workers, CPUs or caches. It implies that communications inside the node are ignored, in particular we will not be able to discriminate between fully MPI job and hybrid OpenMP and MPI jobs.
- Every byte is counted, they can be MPI's communication, data to be read from/written on disks or even control messages. As the bytes are aggregated, the behavior of resulting communication data can be very different of what we expect in communications' analysis in HPC if they do not represent the majority of exchanged data.
- When the network card counts an incoming or outgoing byte, it does not discriminate where the byte came from or where the byte goes. Every incoming byte is counted together no matter where it came from and the same holds for outgoing bytes. Even if communications in HPC sense are dominating, the methods of pattern analysis are not directly applicable.
- The data are aggregated by step of 5 seconds, which is quite long compared to the period (or frequency) of a CPU. The common study of communications patterns is often at a much finer scale [Knüpfer et al., 2008], we expect that many patterns will be aggregated if computational iterations period is less than 5 seconds.

We will still call communication data the collected raw bytes counts even if it doesn't have the same meaning as the classical HPC definition of "raw communication data" when we need to differentiate them from the classical HPC sense.

1.2.3 . The state of monitoring of computing center

For now, we only presented data that are collected but not used currently for monitoring. Monitoring of the computing center is often made with log data. We give the most generic definition

of log data and describe, they are created and we present how it is used practically. Finally, we present the current state of automated processing of such data.

Logs for monitoring

Even with sensors data, it is impossible to observe the exact sequence of instructions and intermediary results *in situ*. To know what a program is currently doing, a developer must in advance plan that the program will print information during its execution and include such instruction in its content.

The event logs are files maintained by the operating software where programs are allowed to write information as text strings about their execution. The event logs are then a way to monitor the execution of a program and record the events about a software by the system which were evaluated as relevant by the developers of the software themselves. That's why they are also one of the main tools used by the computing center administration to monitor the computing center.

Each event is recorded as a string of human-readable text along with the timestamps of its record. The string of text is also called the message. Although the text is human-readable, it is highly dependent on how the software developer format it and fill it with value of interest which makes the text a semi structured content.

Several conventions can be used to allow the exploitation of logs. The developer of a software can allow its user to print different verbosity level depending on a criticality level (ERROR, WARNING, INFO or DEBUG level). The logs can be aggregated along with the name of the programs or Process IDentifier (PID) to improve subset selection of logs against such criteria.

Practical software solution in current HPC centers

Although made for human, aggregated system logs are notoriously hard to read by a human, even with a lot of expertise. One issue is that each developer will use a different log format for his/her software. This means a log can really be interpreted only by a human who knows the software which print such logs.

To help the administration to monitor the computing center, logs are aggregated and parsed to make visual rendering. Information in logs are combined with other data collected, in particular usage data. All aggregated, those data can be used to create alarms with automatic threshold and a human operator can check the status of the system.

The proposed tools for monitoring by the private sector are often an infrastructure to collect some data and metrics on the node usage and aggregate them in a visual display so that the operators can visually monitor the usage of the computing center. Certain known log pattern can be collected to improve event detection. They also provide services like integration and deployment of the material and support.

Some example of such commercial solution are Datadog. Manufacturers like ATOS can also provide tools to monitor their computing nodes.

State-of-the-art monitoring

Several papers have already been published on the monitoring of HPC center in the literature. Many articles focus [Nikitenko et al., 2016] on how to build a data extraction system to aggregate them in a visual manner which is easier to interpret for the human. This is the step before rendering of data of the visualization pipeline, the rendering itself is often a commercial solution or made by the operator of computing center directly. Such rendering is the most important tool for monitoring.

Monitoring is mainly human driven. The best indicator of a problem is a user calling HPC center support line because something went wrong with one of his job. The HPC center operators also try to build a reproducer, a program which goes wrong in the same way, using log files and information from the user on what he tried to do. The reproducer is then used to find where the problem comes from and the operators may add some indicator to detect a similar problem next time if it was unknown before.

These steps are hard to make automatically. The first step to automation of monitoring would be to be able to identify logs that are generated by the same instruction of a program in the aggregated logs file. This problem is called parsing and it is notoriously hard to solve [He et al., 2016] with machine learning, and it is also very difficult to evaluate the quality of the result. [Zhu et al., 2019] made an extensive review of the state of the art of log parsers and present an industrial deployment of parser. They acknowledge that it cannot be reliable without regular human intervention to update rules based preprocessing and control the concept drift introduced by the changes of log structure because of upgrades of the software. This result in a painful maintenance for engineers. Some papers claim to provide good results of event analysis [Klinkenberg et al., 2017] on the monitoring of failure event but rarely describe how the logs were preprocessed to make the analysis. So it is very hard to evaluate if the results are sufficiently good to be useful in practice. Several parsing algorithms based on statistical analysis of the occurrence of words or tokens of a log line [Vaarandi and Pihelgas, 2015] were proposed, but it is very hard to reliably tell if they are accurate enough as preprocessing to not impact too much performance of the monitoring of the sequence of logs as events once parsed as already pointed by [He et al., 2016].

The monitoring of HPC center using logs is as challenging as it is promising. It seems that machine learning has not enough guarantees to be reliably used without efficient preprocessing and choice of analyzed data. This already requires knowing what type of error we want to identify, which somewhat defeats the original purpose of log analysis : discovering new ones.

1.2.4 . Simplified model of extracted data on the computing center

A global representation of the computing center studied in this work and data collected is shown figure 1.1. This sketch does not aim to represent all the part of the computing center with equal details but summaries previous sections about which data was available and used in this work and where it comes from. The elements in the white rectangles are hardware part, the software part is in gray rectangles and the humans are not inside box. The Log collector is described as a software which is often an implementation of syslog protocol, the logs data can be stored in the same server or not.

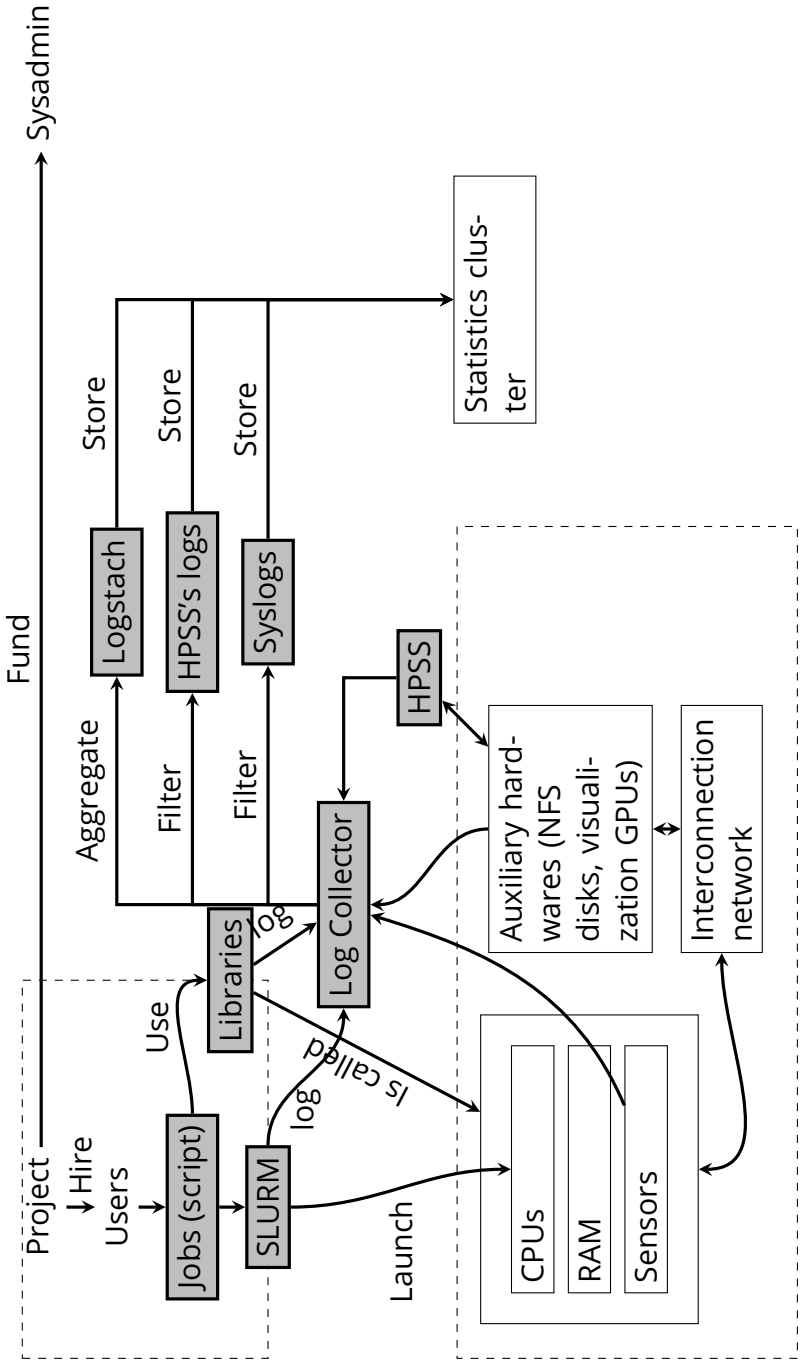
The aim of this work is to propose methods to have a better understanding of this complex system using the data available. As we saw, the data can take many forms and combining them may require expertise that deserve its own research field. Instead of working on the whole system, we focus on solving interesting issues involving only a subpart of the whole system described for which data can be combined.

1.3 . Research tracks proposals for the monitoring of HPC sys-

Figure 1.1 – The computing center parts and related data collected implicated in our research tracks. White rectangles correspond to hardware elements, the different parts of software are shown in gray rectangles. Humans actors or group are shown without box.

The arrows label the information flux and known interaction between elements.

The dotted rectangles are interactions enclosures : we have no reliable data about the interactions between components inside. These are the user data and the direct connection between core hardware components of the computing center.



tem

After describing more formally a computing center as a complex system and the collected data, we now propose some research tracks on the topic of monitoring such system or subpart of it using the statistical analysis of the collected data. We identify 3 issues that are worth digging : the prediction of power consumption of a job using its RJMS submission data, the visualization of aggregated count data at the job or duration level and conception of well-defined summary statistic about the usage of resources and Parallel Programming Model of a job using time series of collected metrics, in particular raw communications.

1.3.1 . Formalization of power consumption prediction

We saw that the main cost in HPC center operation is now the cost of electrical power supply. Most of the HPC center act as a high demanding agent of the electricity market, the power consumption of an HPC can spike because of high demand of users. A prediction of the power consumption of a job may help in monitoring it. We first describe the ideal goal and how such prediction may help a lot to achieve it, then we present what are the relevant assumption for such work to be useful for monitoring and finally add what features we need in the model used.

Monitoring power consumption of using incentives

We saw that the administration of an HPC center must collect some information about the computing resource usage of a user in order to allocate them for him. This work is automatically made by the RJMS that asks the user for the duration of the allocation, the type of computing nodes and how many nodes the user need.

The allocation policy of the RJMS is generally a priority queue combined with back-filling [Mu'alem and Feitelson, 2001]. At any time if the computing center can provide enough computing resource given the demand for it, it will immediately allocate all the asked computing resource no matter the potential spike of power consumption it may create. If all the computing resources are allocated, the RJMS will sort the waiting jobs submissions based on criteria that can be internal, like the duration of the job, the number of nodes to allocate or the number of jobs already submitted by this user, or external, like a priority score on a user defined by the administration of the computing center which may depend of the price paid by this user for example. Back-filling allows the RJMS to satisfy faster a job allocation using their durations to fill the schedule and increase the utilization of computing resources.

The RJMS could monitor the power consumption by using it as a supplementary constraint like computing resource are. It could delay some jobs to smooth the power consumption of the whole computing center in time. The problem is that the RJMS need to know how much a job will consume before it is run, and we already saw that application software run by users are unknown before it is executed.

One solution could be to ask the user to add in job submission send to the RJMS what the consumption of his job will be. But this is a bad idea because even the user may not now how much his application will consume before running it and even in the very ideal case where he knows it, the user estimate is inherently user-malleable, he has incentives to give a strong underestimation instead of a fair value so that his job will be run earlier.

This means the RJMS needs a prediction made by itself to reliably monitor the power consumption of the computing center.

Case where reliable prediction is feasible

The feasibility of monitoring the power consumption of a computing center using the RJMS mainly depends first on the desired level of monitoring and the material available to realize it. Battery can be used to smooth unavoidable sharp transition in utilization but they can also be bigger to monitor a budget of energy fixed by the supply as [Dutot et al., 2017] to make the HPC center a electricity consumer of last resort like some cryptocurrency mining farm can do to get a cheap electricity. The HPC center administration may also prefer to not add any delay in resource allocation and act as demand in electricity market. The needed level will not be the same for all computing centers depending on the priorities of each one and it is out of the scope of this work.

However, it requires having a prediction of a job power consumption before it runs, it is clear that the reliability of the power consumption estimation that the RJMS can make is a very important factor of the feasibility of monitoring through the RJMS. This means that what impacts the reliability of estimation must also be taken into account.

Some computing center will incentivize job redundancy. This is the case of an industrial computing center like COBALT where users paid computing hours in advance and are incentivized to use them no matter what. In such cases, it is clear that the job power consumption estimation will be more reliable in average since the exact same job will be submitted several times by the same user, with slightly different physical parameters. This may not be the case of all computing centers, particularly in the case of on-demand pricing where we might expect user behavior to be less predictable.

We also assume that the administration of the HPC center has a prior knowledge about what type of application users will generally run. This again applies well on an industrial computing center like COBALT. Users often interact with computing center administration and present them what common applications they will run on the computing center before using it. This allows the administration to agglomerate users in group project which can be used to have an idea on how much of each computing resources (CPU, RAM, Inputs/Outputs or communication) they will use in advance. In cloud computing, administration rarely has any interaction with its users directly. So such grouping wouldn't be available in much of the cases.

We focused on the COBALT computing center in a attempt to make a proof-of-concept to know if reliable prediction of global power consumption with statistical models is doable in a expected good case.

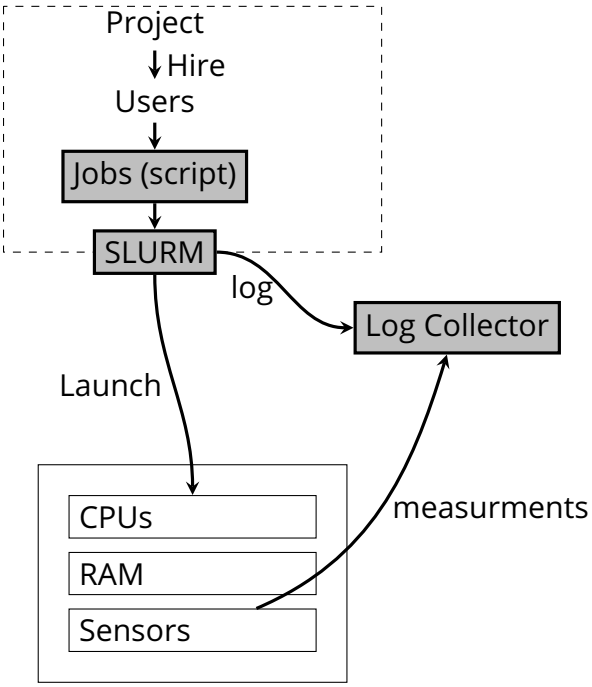
The data, the predicted value and the model constraint

Because we want to use the estimation of the power consumption of a job before it is even scheduled by the RJMS, we must restrict the model to use only the data available when the job's submission is received by the RJMS. These data are composed of what the user must send to the RJMS for his job to be scheduled correctly and the executable location, the RJMS also uses the user identifiers to schedule the job.

Once the job is finished, the RJMS collects aggregated data on the job like the duration and total energy consumption and append the submission data and usage data to a log file that we use to train our model. Naturally, our model must not use the usage data to make its prediction. However, the predicted value can be a function of any value in the data. The framework in which the power consumption prediction must be made is illustrated figure in 1.2.

The data is also collected in a stream in production context, so the model should be fitted online to avoid storing data that may not be useful anymore after a long time. Although we expect usage value to be quite redundant in the short term, we can't expect users to always have the

Figure 1.2 – The part of the global computer center model implicated in the power consumption prediction



same usage. This could introduce "concept drift" [Webb et al., 2016] which would require retraining the model. An online model handles this more naturally and there are several strategies to handle it [Gama et al., 2014] provided the model is simple enough.

We develop our proposal in chapter 2.

1.3.2 . Visualization framework for aggregated high-count data

We identify that having correct representation of high-count data is a major research track for machine learning to be useful to help monitor in industrial context. We first explain how large count data naturally arise from industrial monitoring. Then we show how the HPC data naturally fit in this framework. Then we show what assumptions can be made to get helping visualization and the challenging issues that remain unsolved.

Monitoring and high count data

Complex industrial system composed by humans cannot be monitored only through monitoring of physical quantities. Because human are in the loop, the monitoring is made through events which report what the system is doing in respond to some human command or other events. Because the system is complex, a chain of events more or less critical and relevant may happen and they are logged in some database. The number of events and their diversity in such context can be massive. They are often aggregated so that we can easily group them by a certain criterion when an operator needs to figure out what happen retrospectively. The grouping criteria can be the expected severity of the event, the subpart of the system involved or the date.

The hand made analysis of event data is very hard because it requires expertise about how human are expected to act and knowledge of which events are normal or not in which cases. Even if aggregation of events by counting them removes some information, it can be a simple but efficient approach to visualize, compress or preprocess the dataset before feeding it to a

model for training or prediction. It is then natural or even technically required to aggregate the events to process and store them. Because they are many events in the groups, the resulting data will be a time series or vectors of high counts.

This is how high count data naturally emerges from the event monitoring of complex system.

High count data in HPC monitoring

We formalized the HPC center as a complex system in which humans are a subpart. So there are many events which happen in response to humans commands and other events, and we can deduce many types of high count data.

Such events occur in a sample space (in the probability theory sense) with at least three independent dimensions to aggregate them :

Time aggregation : Events can be grouped together if they happen during the same time period. This period can be variable or not. Regular time windows are used when we want to obtain time series of resource utilization. It is variable when we don't know when the relevant period ends, particularly when we want to understand users' behaviors. This is the case of jobs where the duration of the job is not known in advance. Both aggregation can be combined to obtain time series restricted to a job.

Computing units aggregation : The event happening in the same subpart of the computing center must sometimes be aggregated to avoid that the data acquisition disturbs the computing center utilization. However, some events are collected on the node while they are related to the usage of cores of the node. The reason may be that there is no way to conceptually map event with a core (the shared node's RAM utilization cannot be partitioned in individual core RAM utilization). Another reason is that the sensors measured events at a fixed granularity and finer information is lost (the raw bytes counts are measured by the node's network card which is not aware of bytes exchanged between core inside the node).

Software criteria : The software also create type of events which can be recorded and aggregated. It can be the call of a function or the reporting of a state value or event in a log file. This also mean that we have to define a meaningful event type. The type by which events are aggregated can be a software, a library of functions, a function or a line of code. In the case of log lines for example, logs can be aggregated with respect to the software which produce them, or by the line of the code which produce it. This implies that we must at least be able to find log generated from a common software or even template.

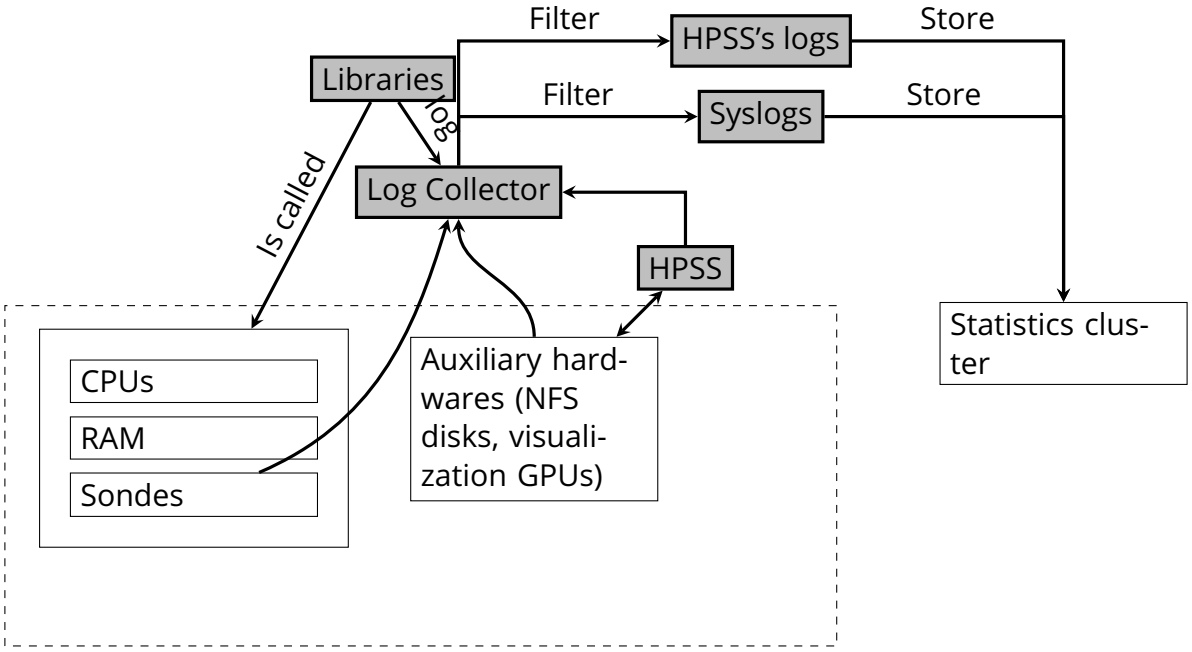
Given all the events' aggregation that can be performed, the statistical analysis of high count data is necessary to better understand the utilization of an HPC computing center.

Use cases of high count data analysis in HPC monitoring

We proposed a proof of concept presented chapter 3 of high count data visualization for monitoring and understanding of computing center usage with log data and using two types of aggregated high event count data to show how useful a model adapted to high count data can be to better monitor the computing center.

We looked at how it can be used for the log analysis which we expect will have a strong impact on the automatic early detection of problems. While it is easy to aggregate log by nodes, time and even software, the main issue is that we need to group log lines more precisely and this is hard to do. We first show that a very simple but efficient visualization of the evolution of the computing

Figure 1.3 – The part of the global computer center model implicated in high count data analysis



center can already be useful if we know how to group them by template, which is however very hard to do reliably. Therefore, we perform high count data visualization on logs data of HPSS subsystem where an identifier of the type of logs is also returned.

We also looked at count data available only once a job end. A great candidate for high count data is the information extracted by light profilers which are added to shared library by the computing center administration. Like profilers, they return information about effective hardware and software usage but aggregated on the whole run of a program to be "light" in the sense that they do not slow the profiled program and often count function calls or CPU cycles.

We summarize which part of the computing center are involved in this research track in figure 1.3. The main issue with such data is the structural zeros which are hard to handle because they are not explained by the low frequency of the event counted. That is also why we will not apply such analysis to time series of counts that tend to have a lot of them.

1.3.3 . Time series analysis of raw communication metric

We finally propose a model of the time series of raw bytes' communication. The model can be used to build new estimators of resource utilization with less bias. We first present how estimation of resource utilization is currently used. Then we show the limit of the currently used estimators and which issue should be solved to improve them. Finally, we present a promising solution on the time series of raw bytes communication that we will be explained in depth in chapter 4.

Calibration of an HPC center with utilization data

We saw that the economic reason HPC center exist at the first place is to share the capital expenditure among users so that they all benefit from more computing power. This means an HPC center must be upgraded to remain economically attractive. While the software are kept updated to the latest version by operators to benefit from their best performance, hardware

must also be changed regularly, either because of failure or because new and better hardware is available on the market. This raises important economic issues which are at the core of the business model of an HPC center.

This means the administration must choose when the hardware is replaced and by what. The replacement has several costs : the price of hardware but also the duration where the computing resources changed won't be available to users so they may not be able to run the programs they usually run. The new hardware must be attractive enough for current users. So the administration needs to know what kind of computing resource is a limiting factor for the HPC center's users.

Knowing the limiting factor of a job is not trivial and a part of the high performance computing research field is dedicated to this task. It often requires running modified codes that return more information on what computational unit are doing, using, receiving and sending called trace. But this reduces the computational performance, so we cannot expect users to run them, and even if they may sometimes do it, we cannot expect them to share what they found because it goes again their confidentiality.

That is why sensors were added on the nodes of COBALT computing center directly to collect more precise usage data without modifications of software run by users and with minimal impact on the computational performance.

Improve utilization understanding with sensors time series

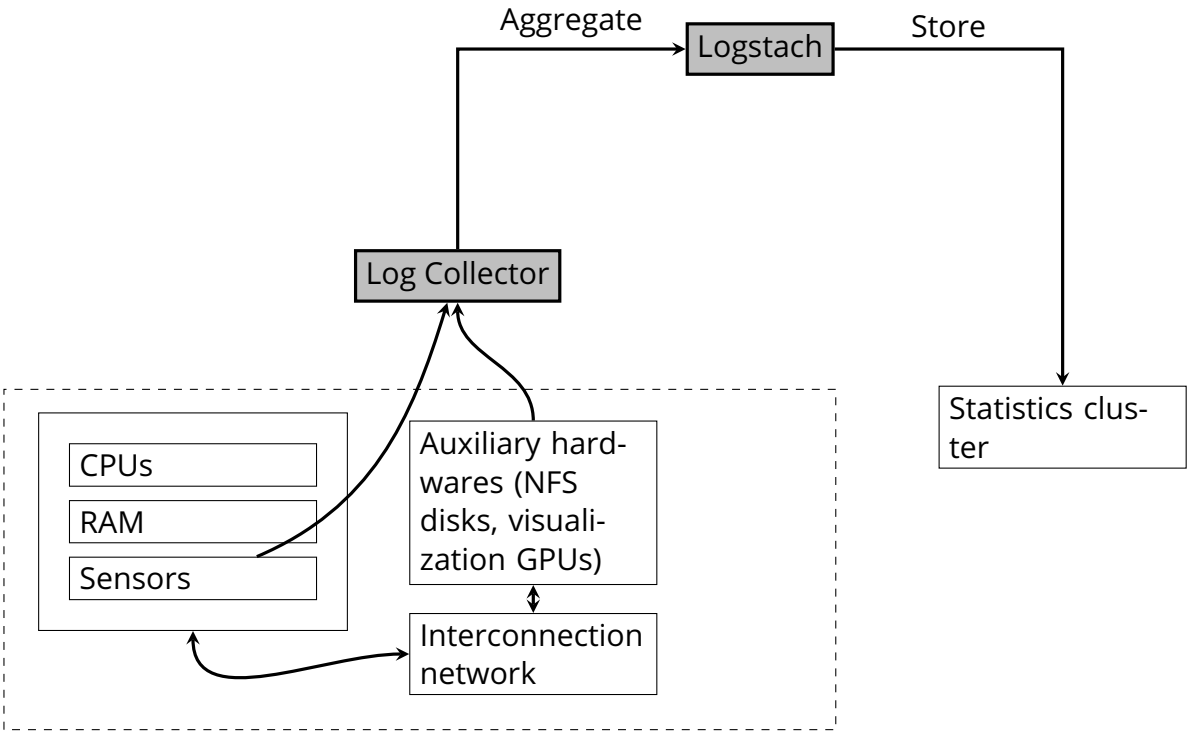
The data collected by sensors can be used to estimate the needs of users with simple estimators. It is generally aggregated by jobs into multivariate time series. The operator will consider certain metric collected and compute some simple estimator on them for each jobs like the minimum, maximum, median or mean. The choice for the new hardware is based on these estimations. If CPU usage of a job is close to 100%, we expect in a first approximation that it shows the job is limited by the CPU, we say it is CPU-bound. The same can be done with RAM usage. Another interesting metric is the amount of communication between nodes or storage disks. If the number of bytes send or received is high then we might expect the job is limited by the bandwidth of the interconnection network (we say it is communication-bound), the same analysis can be applied for the data exchanged with storage disks also called I/O (Inputs/Outputs).

The needs of users are estimated thanks of this rational. But the time series property is never really used in the estimation. This can introduce bias in previous estimators. Indeed, a job is simply the time period during which computing nodes are dedicated to running the user's programs or interactive session with him. This means there are periods where no program is running, or they take time to load to fully use the computing resources. These periods are still taken into account with simple estimator of utilization. It may be useful to automatically detect with time series analysis when the program is performing the most intensive computation to restrict the previous estimators only on the relevant periods in the data and also quantify how much time is not dedicated to the main computation.

Another issue is that different nodes in the same job can perform different tasks or even be used by others at the same time. The previous estimators do not use the patterns that may allow to identify such partition of nodes and this information is lost. Being able to get such estimators by tasks requires clustering the nodes time series, if it is possible, then using previous estimator on each group.

Thus, a systematic clustering and segmentation of the sensors time series would improve previous estimators of the resource utilization by applying them to different groups and periods and avoid mixing all of them.

Figure 1.4 – The part of the global computer center model implicated in sensor data analysis



Classify jobs by usage with raw byte communication data

Our goal is also to determine if the time series of sensor data can also provide useful information for practical monitoring although the reason they are extracted is having a better understanding of HPC center utilization for next HPC center design. Such information could be a method to classify jobs and determine if resource utilization is suboptimal for example.

The full time series is available only once the job is over, we are in an offline learning framework. But we need that our time series preprocessing is fast enough to be run each time a job is over to return estimators that could be used to monitor next jobs. It is difficult to use each time series of the hundreds of metric collected per nodes for all the jobs because the computation of our preprocessing will at least take a linear time in the number of metrics used and it can be difficult to interpret what we are modeling in the nodes' utilization. So we must focus on fewer metrics.

We choose to focus only on the raw bytes' communication data because it is one of the best known metrics to characterize the Parallel Programming Model of a program and a whole field of HPC research is dedicated to study finer pattern in communication as already detailed in section 1.2.2. We find that a clustering of nodes by task and segmentation of intensive computation period is possible using the patterns in the raw bytes' communication, although communication are highly aggregated by nodes and time period.

The involved part of the computing center of this work are presented figure 1.4. We present our work on such preprocessing in chapter 4.

1.4 . Conclusion

We introduce the subject of this thesis : the large-scale computing center. Like a personal computer or PC, a computing center is composed of hardware material running software but contrary to PCs, the human actors are an integrated part of the computing center that must be modeled. This decomposition of an HPC center allow us to provide more context about the data we can collect on it and its potential usage and the current state of its monitoring. Using this simplified model, we propose machine learning research tracks about using collected data to improve the monitoring that are detailed in further chapters of this thesis manuscript, starting with the most interesting monitoring that can be improved : the power consumption of the computing center.

2 - Predicting job power consumption based on RJMS submission data in HPC systems

This chapter is for the most part published in [Saillant et al., 2020].

2.1 . Introduction

2.1.1 . Power consumption, the new limiting factor

The primary material to run a computing center is the electrical energy. The Moore's law is often generalized to the observation that the computational performance of the computing centers increases exponentially. Despite this exponential increase of the production, the electrical power was not the main issue at first : the growth of the HPC facility business was driven by the growth of computations rate of nodes, in particular thanks to the growth of the clock frequency. The added value by the computation speed is so high that it is always more profitable to increase frequency even if electrical power consumption increase. However, this growth of energy consumption is not sustainable and has come up against more and more limits.

The physical limit of the clock frequency growth was reached in 2000 years because of thermal dissipation and quantum physics limitations. The growth of computations rate was still sustained thanks to the growth of the number of transistors in cores of decreasing size and the growth of the number of cores themselves. This implies that the applications was run on more and more cores and the optimization of the computations rate require a fine-gained study of the parallelism behavior of each one. The growth of computation rate by the growth of the number of cores implies that the evolution of electrical power consumption is more and more in linear dependency of the one of computational rate.

This implies that electrical power is now a limiting factor to increase the computation rate of an HPC center. The cost of the primary material, electrical energy, become much more significant. The electrical supply weights today more than half of the cost of the biggest HPC center business model. The time evolution of the electrical power consumption of the largest HPC center since 2005 is shown in Figure 2.1 and illustrates this exponential increase. The power consumption of an HPC center will be a new limit in the HPC computer race as it may reach the order of the production of a thermal power plant (100 MW) for ExaFLOP performance [Bugbee et al., 2017] which implies drastic change of infrastructure and business model because electricity is hard to store.

That's why minimizing electricity consumption to reduce production costs and environmental issues is of ever increasing importance. It is illustrated by the creation of the Green500 ranking of computer systems in 2007, it ranks the 500 most energy-efficient computer systems to raise awareness other performance metrics. There are several way to leverage the power consumption issue. The main one is power efficiency. In a common effort, the manufacturers build more efficient hardware and the HPC developers optimize the applications to reduce their power consumption.

2.1.2 . Monitoring through power-aware scheduling

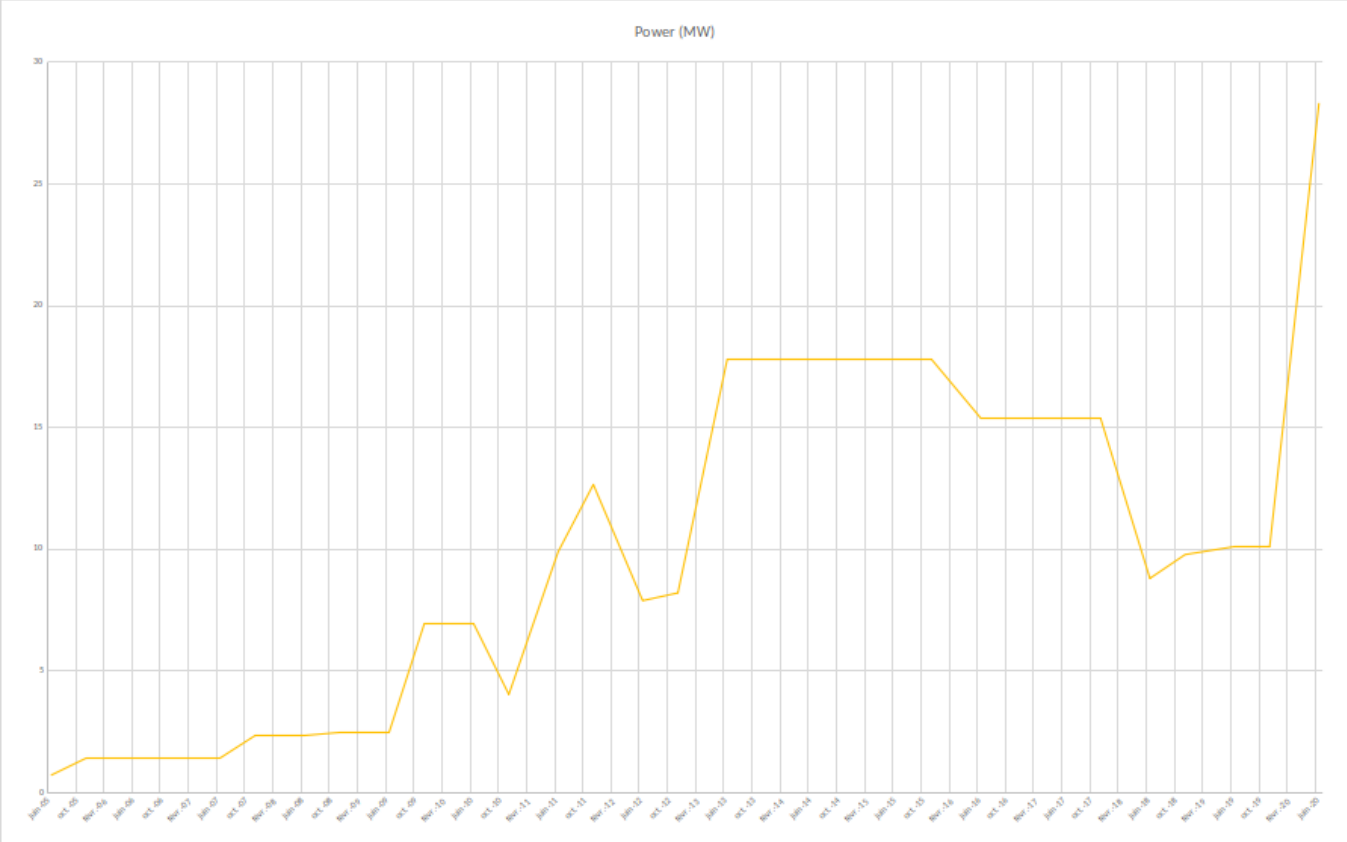


Figure 2.1 – Time evolution of the power consumption in Megawatts (MW) of the first HPC center of TOP500 ranking from June 2005 to June 2020

Another solution is to monitor the resource usage of the computing center. Indeed the instantaneous power consumption of the computing center is not optimized to reduce the energy cost, the users' jobs are run as soon as possible. It implies that the power consumption can spike which increases the charged electricity price for the same total energy consumption. Without monitoring it is also not possible to increase the power consumption when electricity is cheaper, in particular when new renewable electrical production is high and must be consumed.

Resource usage monitoring has several requirements. The power consumption must be first measured at the desired precision and time resolution, this is done with watt meters on each node [Georgiou et al., 2014]. This implies that the measured power consumption is the aggregation of the consumption of the cores of each node. To monitor the consumption we must be able to limit the global instantaneous consumption of the HPC center. The trivial solution is to bound the consumption of the hardware with implemented manufacturers solutions, which generally limit the clock frequency. But it is frustrating for the user to discover that his job is not finished on time because of it, it may also crash the jobs, particularly in a parallel paradigm when nodes may have different clock frequency. A much better solution is to incentive the user to limit himself his own consumption as the HPC center wish. This can be done thanks to a charging policy based on the consumed energy measured by watt meters and the electrical price. However, this radically changes the HPC center business model because it transfers the electrical market price risk from the HPC center to the user.

We are more interested in a solution that doesn't imply such change. The Resources and Jobs Management System (RJMS), or SLURM job's scheduler is the subset of the general structure of the HPC center shown in Figure 1.2. It is the software in the current management of the computing center that have a jobs' monitoring function. It monitors what will run in the short term with respect to a scheduling policy. This scheduling policy could take the consumed power into account to monitor it without significantly changing the habits of users and business model of the HPC Center. The RJMS can then schedule jobs so that the power consumption does not go beyond a budget [Dutot et al., 2017] that may vary when starting a job run or follow any other monitoring policy based on power consumption [Borghesi et al., 2018]. This is called power-aware scheduling. It could avoids usage peaks and progressively reduces the global usage when electricity price are too high and only delay jobs a little.

2.1.3 . Asserting the prediction possibilities

In practice, it is not so easy to implement a such policy because we do not have access to the consumption of a given job before it is at least starting to run. The RJMS monitoring needs a reliable estimation of job power consumption when user submit its jobs so that the RJMS schedule them according to the implemented policy. This estimation framework is challenging, the only data that can be used for inference is the one asked to the user when he submits his/her job to be scheduled and we may suppose that the users are not trustworthy. Figure 2.2 shows the estimation framework. We want to provide an estimate to the RJMS of the power consumption so that it can monitor the power consumption of the HPC center which is the main operational cost. This implies using only the data that are send to the RJMS to infer it.

The use of RJMS data in this type of problem has already been investigated using application types [Bugbee et al., 2017] or symbol information [Yamamoto et al., 2018] to derive a predictions. In these works, the data is not restricted to submission data.

An online model to forecast the elapsed time of the job using only the data given at submission and the current user's usage is proposed by [Gaussier et al., 2015] so that the RJMS can use

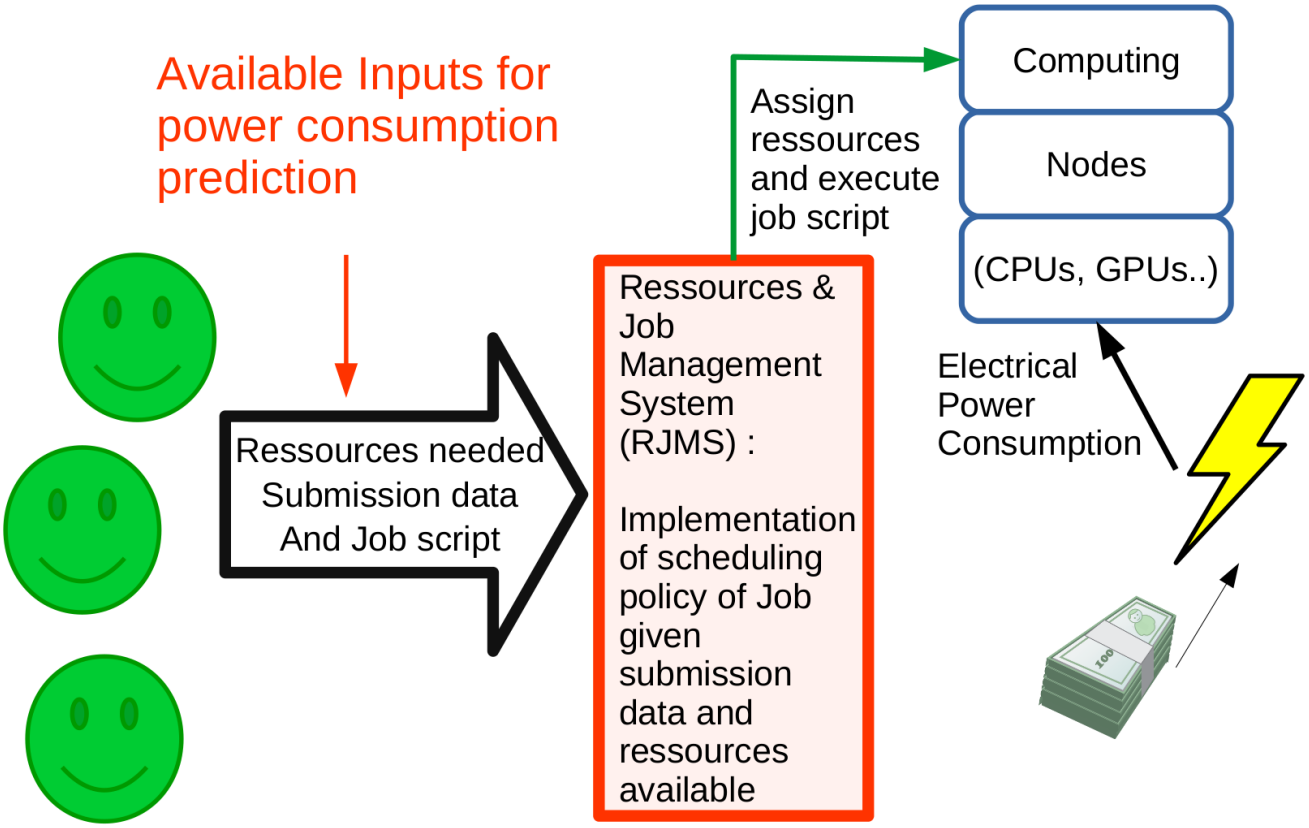


Figure 2.2 – Context of the power consumption modelling.

backfilling more efficiently [Mu'alem and Feitelson, 2001]. This estimation is designed for backfilling and may not be good for power-aware scheduling. An estimate for memory usage and run time using only submission data has been proposed [Tanash et al., 2019]. Although these papers did not estimate power consumption, the used inputs suggest that user information is needed to provide a practical estimate when application types are not available.

Except in a few cases, the instantaneous power consumption of a user and the whole computer center can be predicted with workload information as the number of nodes, components used by the user's jobs and runtime [Sîrbu and Babaoglu, 2016]. However the time evolution of power consumption within a job is not available in the SLURM log files. In a further study [Sîrbu and Babaoglu, 2018], submission data to predict job duration, and not power consumption, are used with the executable name as input data.

Our study consists of assessing the practical feasibility of estimation for the realistic framework for power-aware scheduling purpose by data exploration and a prediction model explainable and usable in production context.

Before proposing a model, we explore the data to define the best prediction target and process the raw data into relevant features. We propose an instance-based model to predict average power consumption using only the submission logs and trusted user provided job data to the SLURM RJMS. We extant this model to production context with online computation which make

it easier to use and maintain in production while a weighted model is introduced to predict the global power consumption of all jobs using instances re-weighting.

Submitted data appears to be sufficient to provide a good estimate of job power consumption for the RJMS. This can be used in power-aware scheduling with our generic model because job submission is redundant. This model may be used in other industrial HPC facilities for power-aware scheduling because it uses only the data that is necessary for scheduling and because it is easy to maintain, but further tests must be made using data from other computing centers.

The chapter is organized as follows : we first extract and pre-process log data from SLURM RJMS [Yoo et al., 2003] and we introduce an instance-based model to process the submitted data as categorical inputs. The model is then adapted to remove biases and to handle streamed data. The final section presents the results.

2.2 . Extracted data and preprocessing

2.2.1 . The COBALT supercomputer and the SLURM RJMS

The data used for this application are collected from the COBALT [1] supercomputer, more precisely, from its main partition which is composed of 1422 nodes ATOS-BULL with Intel Xeon E5-2680V4 2.4 GHZ processors that have 2 CPUs per node and 14 cores per CPU. The Thermal Design Power of each CPU is equal to 120 Watts.

The energy accounting and control mechanisms are implemented within the open-source SLURM [Yoo et al., 2003] Resource and Job Management System (RJMS) [Georgiou et al., 2014]. The data are recorded from accounting per node based on the IMPI measuring interfaces [Georgiou et al., 2014]. IMPI collects data on the consumed power from all the components (e.g. CPU, memory, I/O, ...) of the node, temporally aggregates it and returns the consumed energy during an elapsed time to SLURM. As it is impossible to differentiate between jobs running on the same node, so it was decided to exclude jobs that did not have exclusivity on a node.

The collected dataset is the resulting logs of SLURM submission data of 12476 jobs run on the supercomputer over 3 months at the beginning of 2017 and their respective consumed energy. The jobs that do not have exclusive usage of a node or for which the consumed energy is null are filtered out.

2.2.2 . From raw data to relevant features

There are two potential outputs to predict : the elapsed time and the total consumed energy which are both available once the job is finished. For various reasons (e.g. failure of jobs or sensors), null value can be sometimes returned for energy consumption. Only non-zero values of energy consumption are here considered.

Three groups of information provided to SLURM may be used to predict the output (a summary is provided in Table 2.1)

1. **Information on the user :**

   User Identifier (or UID) is a number identifying the user that submits the job. 200 separate UIDs were observed over the 3 months period.

1. https ://www.top500.org/system/178806

Group Identifier (or GID) characterizes the users that belong to the same company or community sharing the same group. This number allows the inclusion of an *a priori* on what type of job the user runs. 30 unique GIDs were observed over the selected period.

2. **Type of resources required by the job :**

   Quality of Service (QoS) sets the maximum timelimit, and discriminates between test and production jobs.

   Timelimit can be set by the user to benefit from backfilling. This is a continuous system variable, but only 520 distinct values were used over the 3 month period (430 by the same user), showing that users often reuse the same value. Hence, we chose to discretize this variable by taking only the number of hours that are needed (c.f. Table 2.1)

3. **Computing power quantities required by a job :**

   Number of tasks in parallel is defined by SLURM with option -n (e.g. the number of MPI processes if MPI is used)

   Number of cores per task is defined by SLURM with option -c and is used for threading models or if an MPI job needs more memory than is available per core. This information is combined with the number of tasks to form the number of nodes required and is not stored.

   Number of nodes : SLURM combines the number of tasks and the number of cores per task to define the number of nodes needed but the user may specifying this directly.

SLURM logs may also be useful for prediction :

Date of submission of the job. This cannot be used directly as input since no future job will have the same date. However, some features can be computed based on the time of day the submission was made (c.f. Table 2.1).

Final number of nodes that the SLURM allocated for the job. This is the same as the number of nodes required in our data.

Start date of the job can differ from the submission date if the job has to wait to be run, but it is set by SLURM and not the user, so it is not used. The same holds for the **end date**.

Executable name could be used in some cases to identify the type of application the job is running. However, it can be irrelevant ('python' or 'a.out' are extreme examples) and users may take advantage to manipulate SLURM if it is used to define scheduling policy. Hence, it was decided to ignore this in our model.

Table 2.1 summarizes the model's inputs and the potential outputs of interest. Although the number of cores per task is unavailable as it is not memorized by SLURM, it is an interesting value. The average number of tasks per node (tasks/node in Table 2.1) can be computed as an equivalent quantity. Redundant features, such as the required number of nodes that is given by the SLURM RJMS but is almost always equal to the final number of nodes, are removed.

Table 2.1 – Synthesis of the relevant handcrafted input features and outputs of SLURM for the studied model.

Feature	Meaning	Comment
	Potential raw Inputs	Information before allocation
UID	User IDentifier	Anonymized and unique identifier
GID	Group IDentifier	Project membership identifier
QoS	Quality of Service	Indicates if job is in test/production
#nodes	Number of nodes allocated	Redundant with requested number
#tasks	Number of tasks in parallel	E.g. number of MPI processes
submit	Date of submission by the user	Cannot be used directly
timelim	Time limit before a job is killed	Cannot be used directly
	Computed features	Knowledge incorporation
tasks/node	Number of tasks per node	Manually created features
submit_h	Hour of submission in the day	Relevant information from submit
timelim_h	Limit duration in hours	Relevant information from timelim
	Outputs	Given after the job is finished
elapsed	Time elapsed	True duration of the job (wall time)
energy	Total consumed energy by the job	Aggregate temporally and by nodes
	Target	Model output
meanpow	Average power consumption per node	computed as equation (2.1)

2.2.3 . Target and problem formalization

The elapsed time and the power consumption are the two unknown values needed before the job runs to improve the management of power consumption by the RJMS.

The elapsed time inference which has been a subject of interest in several papers [Mu'alem and Feitelson, 2001, Gaussier et al., 2015, Sîrbu and Babaoglu, 2018] improves the backfilling of the scheduling policy so that resource usage is maximal at any time.

The energy usage value returned by SLURM is the total consumed energy used by all the nodes for the entire job duration. The energy consumption increases by definition if the elapsed time increases or if the number of nodes that a job uses increases. The total energy grows approximately linearly with the number of nodes and elapsed time. However, this assumption has some limitations, as it implicitly means that the power consumption remains constant when the job is running and each node uses the same amount of resources over time. Although this assumption is strong, it is not far from reality for the majority of jobs, as shown by [Borghesi et al., 2016], and it can be removed only with time-evolving data inside jobs that is not available.

If the number of nodes and elapsed time are not provided, the meaningful consumption statistic able to be predicted given the information collected by SLURM is the average power per node. The average power per node, denoted by meanpow, is defined and computed as :

$$\text{meanpow} = \frac{\text{energy}}{\text{elapsed} \times \#\text{nodes}} \tag{2.1}$$

Once a model returns the average power per node, the job's power consumption can be computed by multiplying it by the number of nodes and used in a monitoring policy as the estimation $\tilde{P}_{comp}$ for budget control [Dutot et al., 2017] or powercapping. If the elapsed time is given (by other models like [Gaussier et al., 2015]), the consumed energy can also be predicted under the

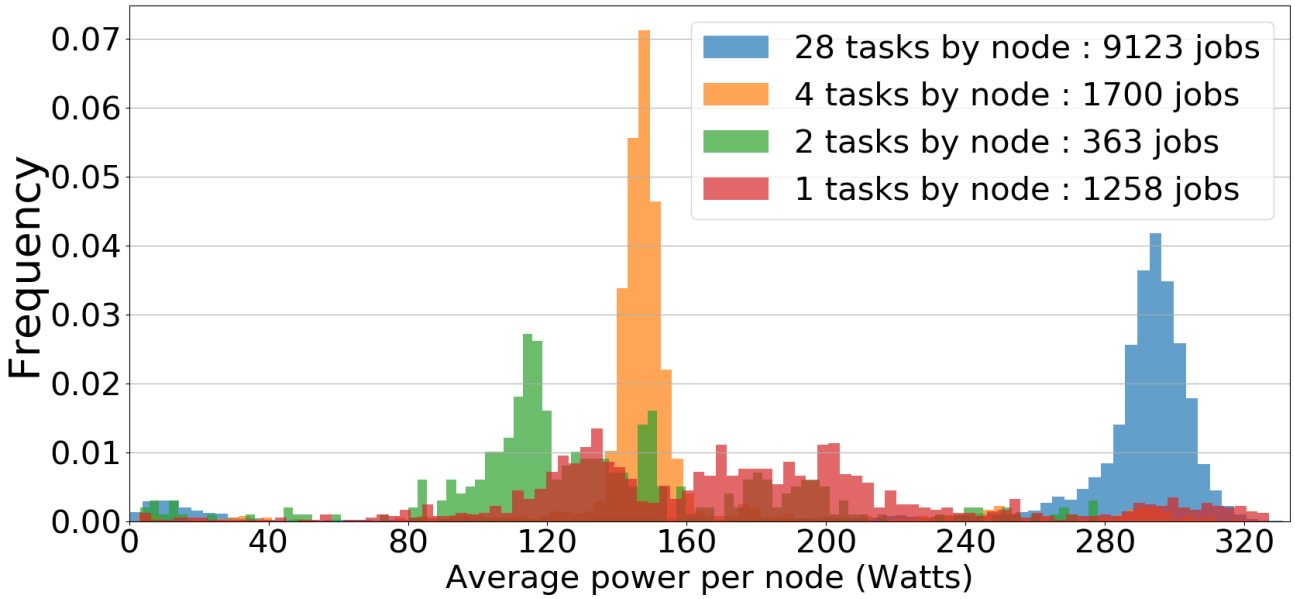


Figure 2.3 – Distributions of average power per node. Each histogram is computed for jobs using the same number of tasks by node. Full MPI jobs use 28 tasks by node.

linearity assumption.

Most of the previously proposed methods use standard machine learning models from the SciKit-Learn python library [Pedregosa et al., 2011], such as decision tree, random forest for [Bugbee et al., 2017] or SVR for [Sîrbu and Babaoglu, 2016]. Those models provide interesting results, but all of these rely on several parameters known to be difficult to tune and they assume regularity in the input space that may not exist in our case. Our first motivation and our contribution are to propose an alternative model that requires fewer assumptions and that works at the same time efficiently.

2.3 . Instance based Regression Model

2.3.1 . Inputs as categorical data

Using all the available data, Figure 2.3 shows four empirical distributions of the approximated average power per node for jobs with the same number of cores per task. We observe that the distributions are well separated with respect to the power. It shows that the number of cores per task is already an efficient criterion to estimate average power per node for certain jobs. This is particularly the case for the most power consuming jobs (reaching 300 Watts/nodes), which most likely use one core for each task, and those using 7 cores per task, which mainly use half of the full power. This is in fact expected as the number of cores assigned to the same tasks generally depends on the threading model of the application, which implies a different power consumption. The Gaussian like distributions contain interesting and useful information. Hence we infer that a low complexity model may be useful for modeling. Combined with other inputs, such as UIDs, we expect a good discrimination of power usage for any submitted job can be made based on a few internal parameters.

In our application, the input features are either categorical or numerical, for example :

— The metadata related to the chosen QoS, the user and group identifiers (UID, GID) are categorical and thus their values (numerical or not) cannot be ordered.

— Other features are numerical and describe two types of information ; discrete (the number of nodes or tasks) or continuous related to date or time (submit, start, end date of the job, duration and timelimit).

However, the discrete numerical features (number of nodes, tasks, or their ratio) may also be considered as categorical variables. For example, an application's performance depends on the number of cores and is sometimes optimal when the number of cores verifies arithmetic properties, e.g. LULESH 2.0 should be used with a number of MPI processes that is a perfect cube [Karlin et al., 2013]. An application running with $27 = 3^3$ cores is likely to be different to a plausible OpenMP application using 28 cores (28 is the number of cores on a COBALT's node). Full MPI jobs use one task for each core while full OpenMP jobs use one task for the whole node. It shows that the threading model imposes the number of tasks per node.

It then seems more relevant to consider the number of nodes or tasks as a class of job or a category. Only 55 unique values were observed for requested nodes in the data when the range of possibilities is theoretically $\simeq 1000$, which confirms the discrete and categorical behavior of the number of nodes or tasks. Although time related data is continuous by nature, we choose to discretize it at an hour level to have categories.

In the end, we transform all available inputs as categorical. We then propose a data-model to predict the average power consumption per node (meanpow) of any job, using only categorical inputs.

2.3.2 . An input-conditioning model

Algorithm 1 instance-based model Training

instance-based model Training

Require: $FeatSelected, Estimator, trainset$
  $Values \leftarrow Unique(FeatSelected(trainset))$ ▷ $FeatSelected$'s values in $trainset$
  for all $val \in Values$ **do** ▷ Group by jobs' value of $FeatSelected$
    $\mathcal{J}_{val} \leftarrow \emptyset$
    for $j \in trainset$ **do** ▷ $\mathcal{J}_{val}$ = Jobs where $FeatSelected$ match val
      if $FeatSelected(\{j\}) = val$ **then**
        $\mathcal{J}_{val} = \mathcal{J}_{val} \cup \{j\}$
      end if
    end for
    $OutputDict(val) \leftarrow Estimator(\mathcal{J}_{val})$ ▷ Estimate output value for val input
  end for
  return $OutputDict$

Algorithm 2 instance-based model Prediction

instance-based model Prediction

Require: $FeatSelected, OutputDict, job$
  return $OutputDict(FeatSelected(\{job\}))$ ▷ $OutputDict$ for $FeatSelected$ of job

Categorical data is generally hard to handle in machine learning because all possible combinations of input values must be considered for optimization and this grows exponentially with the number of inputs. However, though a large number of combinations are possible, only a few are observed in our dataset. Submissions may be redundant and this is a motivation to use an instance-based regression model.

An instance-based model computes a prediction by searching comparable instances in a historical training set. The simplest case of instance-based learning is Rote-Learning [Russell and , 2010] as the nearest neighbor approach with a trivial distance [Cover and Hart, 1967]. The prediction for a given job is an estimator computed on the subset of the training instances that share some inputs as already proposed in [Sîrbu and Babaoglu, 2018].

Let $\mathcal{J}$ denotes our job training dataset. Each job $j \in \mathcal{J}$ is a combination of observed values for the features described in Table 2.1. Rote-Learning is a supervised problem for data as $(X_j, Y_j)_{j\in\mathcal{J}}$. X_j is the vector containing selected input features (i.e. a subset of the submission data as seen by SLURM) of job j used to predict Y_j. X_j is referred as the "job profile". Y_j is the target output of job j computed with any available features. In our case, it is the average power per node called meanpow as defined by (2.1). This is a regression task of Y_j given X_j since Y_j is real valued.

Common regression models make assumptions regarding the behavior of Y given X through a linear hypothesis or a kernel method like SVR in SciKit-Learn [Pedregosa et al., 2011] and assume implicitly that X is either a continuous or a binary variable. In our case X is discrete and these models can be used consistently with “dummy indicators” for each possible modality of a categorical variable. However, the input space dimension grows at the rate of the number of unique values for categories, which makes these models impractical.

On the contrary, the Rote-Learning regression model computes an estimator of the target for the jobs in the training set that have the same job profile as described in the pseudo-code 1 for training and 2 for prediction. We introduce two functions to tune how the predictions are computed. $FeatSelected()$ is a function that extracts job profiles $(X_j)_{j\in\mathcal{J}}$ that are the values from a fixed subset of inputs from job set $\mathcal{J}$. $Estimator()$ is a function that takes a list of jobs with the same profile X_j and computes a chosen estimator as prediction. After training, we return $OutputDict()$ as a mapping or dictionary that returns the prediction of any job j having the profile X_j extracted with $FeatSelected()$. When the job has a profile X_j that is not found in the training set, a prediction for a subset of the profile X_j can be made by another Rote-Learner to handle this case or a default value can be returned.

It is well-known that the Rote-Learner is the best unbiased estimator as discussed by [Mitchell, 1980]. This means that if no prior knowledge is incorporated into another model, the Rote-learner has a smaller loss. Despite this strength, the Rote-learner is rarely used in machine learning because of memory issues and for statistical reasons : the number of samples for each combination of inputs must be sufficiently large and this is rarely the case.

In our case, the number of unique observed inputs is limited in our dataset because the number of samples for a combination of inputs is large enough, hence there are no memory issues. Training time is then short because it is basically the time taken to compute the $Estimator()$ multiplied by the number of unique job profiles in the training set.

In the sequel, we use the arithmetic mean of average power per node as $Estimator()$. This minimizes the Root Mean Square Error (RMSE) for average power per node, which is then the loss function used to evaluate the possible models. In our framework, $Estimator()$ is defined as :

$$OutputDict(val) = Estimator(\mathcal{J}_{val}) = \frac{1}{\text{card}\,\mathcal{J}_{val}} \sum_{j \in \mathcal{J}_{val}} \frac{\text{energy}_j}{\text{elapsed}_j \times \#\text{nodes}_j}$$

with $\mathcal{J}_{val} = \{j \in \mathcal{J} | FeatSelected(\{j\}) = X_j = val\}$ for the job profile val.

2.3.3 . Variable selection

The number of internal parameters that the Rote-Learner has to learn during training is the number of unique job profiles in the training dataset. For a fixed training dataset, this number depends of the choice of subset of inputs that defines the job profile. If this number increases, the model complexity also increases. The model complexity is a statistical concept that quantifies the ability of the model to fit complex phenomena, even in the case of simple noise. But a low complexity model is able to generalize for new data. For this reason, complexity and then the job profile $FeatSelected()$ definition must be carefully chosen.

In our application there are less than 10 features, so the number of possible input feature combinations needed to define $FeatSelected()$ is quite low, and we can exhaustively test all the features subsets one by one and retain only the best one. In this work, a cross-validation procedure is used to find the best inputs : we split our data into two parts, the training set is the first two months of data and the test set is the last month.

This procedure allows the identification of SLURM information pieces which are meaningful. However the computations can be time consuming. Once we empirically find the best inputs, we use only these for the following models as job profiles without repeating the process of finding the most relevant inputs. We discuss the performance results in the experimental part of Section 5.

At its best, the resulting model predicts the average power consumption per node of any job. Nevertheless, this objective is not monitoring the global power consumption. Certain jobs matter more than others and they are presented one by one in practice with no training time, which motivates the improvements of the following section.

2.4 . Global consumption practical estimation

2.4.1 . Weighted estimator for global power estimation

In practice, jobs that run for the longest on many nodes contribute the most to the global power consumption of a computer center. Moreover, we observe a correlation between the duration of the jobs and the average power per node. The scatter plot in Figure 2.4 shows that jobs running for less than one minute consumed less power than the others. A possible reason is that the jobs are first setting up parallelism and reading data from disks. This phase is not generally compressible and does not consume significant amounts of power. If a job is short (test, debug job or crashed), perhaps less than one minute, this phase becomes non-negligible and may then lower the average power consumption, which explains the observed bias.

However, each job has the same contribution to the mean estimate used in the previous section. Short jobs disproportionately lower the mean estimate that is defined in Section 2.3.2, despite their limited contribution to the global consumption. This is why jobs should be weighted by their total consumed node-time (number of nodes multiplied by elapsed time) when computing the mean for global consumption estimation. One method is to sum the total consumed energies then divide by the sum of their total node-time instead of dividing consumed energies individually then taking the mean. More formally, $Estimator()$ must be chosen as :

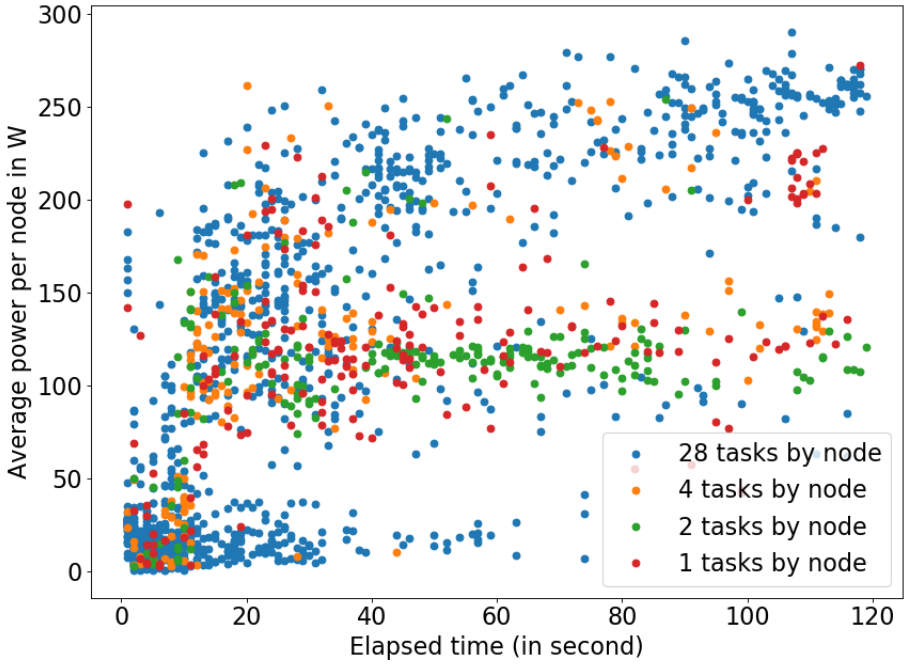


Figure 2.4 – Scatter plot of jobs less than two minutes long, short jobs consume less power.

$$Estimator(\mathcal{J}) = \frac{\sum_{j\in\mathcal{J}} \text{energy}_j}{\sum_{j\in\mathcal{J}} \text{elapsed}_j \times \#\text{nodes}_j} \tag{2.2}$$

2.4.2 . Online computations

Previous section has presented a model which computes offline the estimation of the arithmetic mean : training and prediction are two distinct and successive steps. Once training is done, the model is fixed and used to predict the power consumption of the job.

In the case of job scheduling, data is presented to SLURM as a stream of logs containing information on submitted jobs and the previous two step approach has a major flaw. A model used for prediction does not continue to learn : for a job's profile that was not present in the training data, it can only return a default value at best every time it appears. The whole model can be regularly retrained but it is then necessary to memorize all the recent data seen by SLURM in prevision of the next training round. This approach has other drawbacks : if the rounds are too frequent, a training set may be too small and if they are too rare, a lot of data has to be memorized and the prediction may be worse before the rounds.

Thankfully, the arithmetic mean used as $Estimator()$ can be straightforward to compute online and lots of approaches exist in the literature [Hunter, 1986].

If $OutputDict(X_i)_m$ is the mean estimator at the $(m+1)^{th}$ occurrence of a job with inputs X_i and average power per node Y_i, it can be updated independently once the job is finished as $OutputDict(X_i)_{m+1} = \frac{m}{m+1} OutputDict(X_i)_m + \frac{1}{m+1} Y_i$. The counter m and current value $OutputDict(X_i)_m$ only should be maintained to compute the next value when a job ends. This is called a cumulative moving average or CUMSUM [Hunter, 1986]. This is referred to as an online model because the model is continuously training itself, and a training round is not required.

However, this CUMSUM model gives equal weight to old and recent observations of a job

profile X_i, and thus the expected job average power per node is expected to always be the same. This is not always true : a group of users may suddenly change the applications they use which may impact the power consumption. A good way to account for such a trend-shift is to compute a moving average defined as a mean of recent data within a time-window [Hunter, 1986]. Once again, memorization of recent data is required. However, we need to set the number of recent observations used to compute the moving average, this may not be easy.

An Exponentially Weighted Moving Average (EWMA) introduced in [Holt, 2004] and [Hunter, 1986] for time series analysis is an nice way to compute a weighted moving average without memorizing any recent data. This method weights recent data more heavily than old data according to an exponential decay and then computes the mean. The exponential decay allows the moving average value to be updated with a simple formula :

$$OutputDict(X_i)_{m+1} = \alpha OutputDict(X_i)_m + (1-\alpha)Y_i \tag{2.3}$$

where $\alpha \in [0,1]$ is a hyperparameter to be chosen (values approaching 0 indicate lesser influence from the past). A custom weighting is required to remove underestimation of the computed estimator for global power consumption estimation.

2.4.3 . Exponential smoothing for weighted and streamed update

As stated before, EWMA has the big advantage of memoryless updating but it must be weighted in the update formula (2.3) for global power consumption estimation. Previous online estimators were initially designed and used for time series analysis [Holt, 2004, Hunter, 1986]. To weight them consistently, as in Section 2.4.1 and keep them online, we formally define their associated time series and modify it slightly.

At any time, the value of $OutputDict(X_i)$ is the last estimation of meanpow for a job with the profile X_i since a job with profile X_i ended. The value of the estimate $OutputDict(X_i)$ changes only when a job with profile X_i ends. As it is an evolving mean, it behaves like a trend estimation of the series of meanpow of jobs with profile X_i ordered by end date. Each job contributes in the same way to the future estimation. The contribution to the estimation of a job with profile X_i depends only on which rank it ends. The job's contribution to the online estimation does not depend on its node-time contrary to section 2.4.1. An example of the series and its estimation by EMWA are given in Figure 2.5. We observe that the EWMA is lowered by the low node-time jobs with low meanpow that have the same weight the highest node-time jobs because the series is agnostic to this quantity.

We propose a novel way to account for the needed weighting of the job without changing much our online estimation. The idea is to generalize and compute trend estimate on another series that is irregular. It is the same previous series of the average power per node of jobs with given profile X_i ordered by end date but the intervals between two successive finished jobs is the node-time of the first job, as if a job must wait the node-time of the last before starting. The resulting estimator is a continuous smoothing of this irregular time series parametrized by a node-time constant and can be used as before for online estimation but jobs with lowest node-time will not change the trend estimation as much as the ones with highest node-time. The adaptation of CUMSUM replaces m by the sum of the previous jobs node-time with inputs X_i, and in the moving average case the recent job are weighted by their node-time for example. The irregular series deduced from the previous example are given in Figure 2.5. The short jobs have almost no influence on the current re-weighted estimate even if their meanpow value are extreme. On the contrary, it is clear that the classical EWMA strongly underestimate the irregular series meanpow because of them.

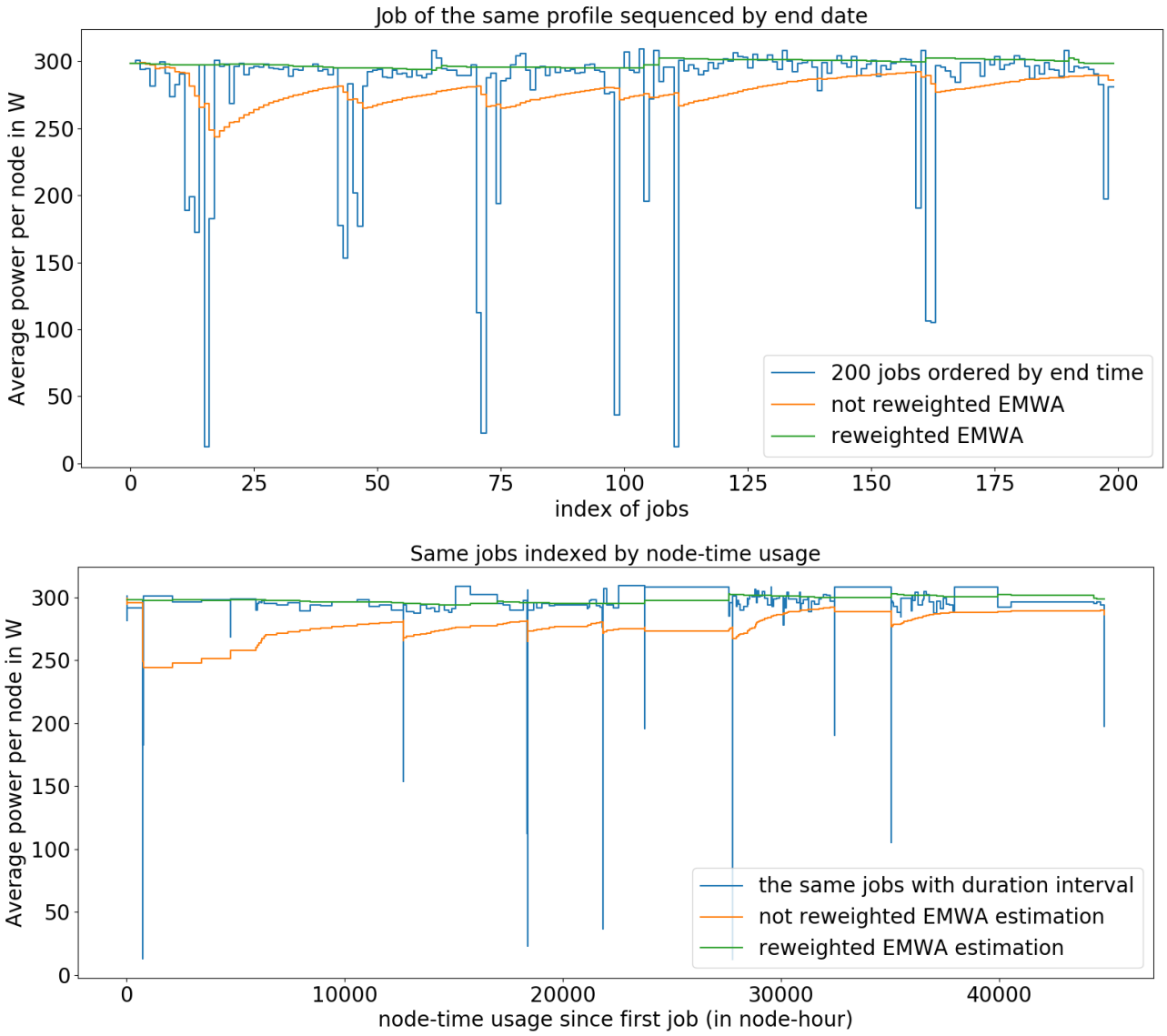


Figure 2.5 – A sequence of average power per node of job with the same profile and the series of its estimations by EWMA with and without reweighting by node-time.
Up : Series of the average power per node of 200 jobs with the same profile and its estimation by classical and reweight EWMA.
Bottom : The associated irregular series used to weight the jobs according to node-time and the same EMWA estimations series.
EWMA hyperparameters are $\alpha = \exp(\log(0.5)/20)$ (a job contribution is halved after the 20 next ended jobs) in regular case and $\tau = 4000$ node-hour in irregular case.

We propose to apply this adaptation to EWMA so that our estimation is memoryless and weighted correctly. We compute directly the weighted estimator without computing the irregular series by slightly modifying the previous estimator formula (2.3) to take into account of the node-time of the current ending job. EWMA is generalized as exponential smoothing and computed for irregular time series in [Zumbach and Müller, 2001] or [Eckner,], the update formula uses variable α to account for the irregular time interval thanks to the memoryless property of exponential :

$$\mathrm{EMWA}(Y)_{t_n} = e^{-\Delta t_n/\tau}\mathrm{EWMA}(Y)_{t_{n-1}} + (1 - e^{-\Delta t_n/\tau})Y_{n-1} \tag{2.4}$$

t_n is the time of the n^{th} sample, $\Delta t_n = t_n - t_{n-1}$ the length of the $n-1$ interval between samples, and τ a chosen time constant of exponential decay.

Applied to the trend estimation of the irregular time series, (2.4) formula shows that α in (2.3) must be replaced by $e^{-\Delta t_i/\tau}$ to weight the job i according to its node-time Δt_i. τ must be in the order of expected node-time value for several meaningful jobs. Due to (2.4), it is not necessary to compute and maintain the irregular time series, (2.4) is used when a job ends and the current estimation $OutputDict(X_i)$ is updated by computing node-time $\Delta t_i = \mathrm{elapsed}_i \times \#\mathrm{nodes}_i$ and setting $\alpha = e^{-\Delta t_i/\tau}$ in (2.3). Our method benefits from both the advantages of EWMA and the weighting correction for global power estimation. Benefits of our approach are discussed in the last experiment of the next section.

2.5 . Numerical results and discussion

2.5.1 . Offline instance-based model

The proceeding described in Section 2.3.3 is run using the instance-based model offline introduced in Section 2.3.2 to determine the best job profile X_i. For each possible job profile, the model is trained using a training set containing the 8000 jobs of the first two months. Then the RMSE is computed for a testing set of 4000 futur jobs from the next month.

It appears that a significant part of the jobs in the testing set show a combination of inputs that were never observed during training. In this case the model does not return an output value if the job profile is not seen previously in the training set. For a fair evaluation of any choice of job profile, we need to avoid handling the case where a pretrained model does not return an output because the profile is not known by the model. For that, we first extract a small test subset from the initial testing set composed of 1022 jobs for which their profiles are present in the training set. This is so that any model returns an output value no matter what job profile it uses.

We illustrate the bias-variance trade-off by showing the best choice of job profile that has a given number length with the lowest score (here the RMSE) and the number of unique job profile values observed in training set is shown as "diversity". Diversity is a simple way to approximate the complexity of the model to highlight bias-variance trade-off. We present the results for the small testing set in the first column of Table 2.2. In the special case where the job profile has zero inputs, the model always returns the mean of the average power per node of all jobs in the training set.

The best prediction requires features that identify the user, because users tend to submit the same jobs. UID is first chosen but the number of tasks per node improves power estimation and is more general when combined with GID instead of UID. Indeed, diversity is lower when the model uses GID instead of UID, and GID still indicates well enough that the jobs may be similar as part

Table 2.2 – Variable selection by cross-validation and results. The score is the RMSE (lower is better). Diversity is the number of memorized instances after training.

	Results on small test			Results on large test	
#	Best combination	Score	Diversity	Best combination	Score
0	(returns the mean)	78.04	1	(returns the mean)	89.87
1	UID	46.17	$\simeq 150$	UID	44.85
2	GID, task/node	43.98	48	GID, task/node	**43.31**
3	GID, task/node, timelim_h	**43.63**	217	GID, task/node, QoS	43.83
4	Add QoS	43.78	232	Add timelim_h	44.16
5	GID, QoS, #nodes, #tasks	45.09	245	Add UID (all features)	45.49
6	Add timelim_h	45.53	475	(no more features)	-
7	Add UID (All but submit_h)	47.55	578	(no more features)	-
8	All features	52.84	1981	(no more features)	-

of the same project given they have the same number of tasks per node. Surprisingly, adding the hour part of timelimit improves the results although it drastically increases the diversity. However, adding more inputs to the job profiles worsens the results, and the effect of over-fitting is stronger as diversity increases. In particular, the number of nodes and tasks by themselves seem to be not relevant for prediction of the power consumption as these parameters are always selected together. QoS does not seem to be informative on power usage. The hour of submission is the last selected feature showing that the type of job is the same no matter what the hour in the day is, which can be explained by auto-submissions.

The number of nodes, tasks and the submitted hour can have a large range of unique values that substantially increase the diversity which means they tend to produce over-fitting. In our experiment, this is observed when the result does not improve if these values are accounted for. But the reduced testing set is constructed only with jobs that have a combination of all these inputs values in the training set. To get more robust results about other choices of input features we reduce the space of possible job profiles which increases the number of jobs in the testing set with a profile in the training set. As these parameters seem not to be relevant for prediction, they are removed, and a larger testing set of 2216 jobs is constructed with the combination of inputs without these omitted values. The results are given in Table 2.2 in the second column.

The RMSEs are of the same order of magnitude as they do not depend on the size of the dataset. The same behavior in variable selection is observed, except that timelimit is no longer relevant, even selected after the QoS (and we can only choose up to 5 features as the others are removed). This difference may be explained by the strong selection of which jobs are included in the small testing set slightly favoring over-fitting.

From these observations we conclude that the GID and the ratio of number of tasks and nodes are the best choice of features to predict the average power consumption per node with any model for data on COBALT. The resulting model of this choice of features will be called IBmodel for Instance-Based model in the next sections.

2.5.2 . Comparison with the offline IBmodel

We compare the IBmodel with models currently in use and proposed by [Bugbee et al., 2017] and [Tanash et al., 2019]. We focus on models based on trees, Decision Tree Regression (DTR) and Random Forest (RF), that are well-known to handle better inputs from categorical features. We

Table 2.3 – Comparison with other models on test set. Score is RMSE (lower is better).
Score (all) : result with all the input features for the large test set.
Score (selected) : results with input features being GID and task/node.
As RF training is not deterministic, it is run 100 times, then the mean score and standard deviation are given.

SciKit models	Tested parameters	Score (all)	Score (selected)
DTR	pure leaves, MSE criterion	48.80	43.31
RF	pure leaves, 0 to 50 trees	(0.15) 45.79	**(0.07) 43.14**
GBRT	max-depth 5, 0 to 300 trees	44.40	**43.10**
SVR	rbf kernel, C=1000, $\gamma = 0.01$	53.58	45.79
IBmodel	X_i = (GID_i, $\mathrm{task/node}_i$)	**43.31**	43.31

also add the Gradient Boosted Regression Trees (GBRT). For these last two models, we increment the number of tree estimators and retain only the best results. We also compare the IBmodel with results from Support Vector Regression (SVR), as used by [Sîrbu and Babaoglu, 2018], choosing the best SVR parameters by manual tuning. The SciKit-Learn library [Pedregosa et al., 2011] is used to run and train the models.

At this stage, the IBmodel is not designed to return an output in case of unknown job profile. So our tests are run on the large testing set of 2216 jobs previously selected and we drop the same features (number of nodes, tasks and the submitted hour) to avoid unknown job profile during testing. In a first test run, all the features used to obtain the second columns of Table 2.2 are the model inputs. In a second test run, the chosen features are only GIDs and task/node, which are the best choices for average power per node prediction found with the IBmodel (hence it keeps same score). We point out to the reader that this favours competing models, especially the ones based on trees for which only the best is retained.

Table 2.3 presents the results with a range of parameters. It is observed that the IBmodel outperforms all other models when the input space is large. This underlines that there may be no variable selection in the other models. However, RF and GBRT outperform when we explicitly force the selection of the relevant inputs we have computed previously for all the models. DTR also provides the same results as the IBmodel as it becomes similar to an instance-based model when the dimension is low and the decision tree's leaves may be pure (they can have only one sample in training).

The interpretability of the instance-based models, in particular the selection of explicit features, is a strong advantage as this improves also the other models. Although RF and GBRT are the best performers once the most effective inputs for prediction are known, we do not think they are the most suitable for our application. First, the IBmodel can be updated online whereas RF and GBRT must be completely and regularly retrained in order to handle a job's stream, second, the IBmodel can weight the observations of average power per node of jobs with minimal modification, and finally, it is not easy to explain how RF and GBRT built their predictions.

2.5.3 . Online IBmodel

For practical monitoring of global power consumption through the RJMS it is necessary to provide online and instance weighted models, as already presented in Section 4. To demonstrate this claim we compare predictions of future global power consumption available at the submission date of a job by the IBmodel with and without these two improvements. We first construct a re-

ference target with an oracle estimation over time. At any time t, the oracle value is the sum over all running jobs at time t of the jobs' average power consumption. The oracle value is not the true global power consumption as a job's consumption can vary when running, but it is the best approximation following the hypothesis made in [Dutot et al., 2017] and section 2.3 (consumption is constant over the job's entire duration).

With the data from the same period of two months used in the previous section, the IBmodel is trained with only improved weighting (2.2), only online updating (2.3) with $\alpha = \exp(\log(0.5)/20)$ (job's contribution is divided by 2 after the 20 next ended jobs with the same profile) and also both (2.4) with $\tau = 4000$ node hours, then the results are compared to the oracle value of the test set. To compute the estimated value of global power consumption available to SLURM, the list of jobs is converted to a list of events ordered by their time t of three types :

- **Submission event :** The job j is submitted at time t, its average power consumption per node is estimated with the models and buffered for a future event. If models cannot return an estimation (unknown input values), we return the default value 292.89 Watts per node as it is the global average power per node of all the jobs we have.
- **Starting event :** The job j starts at time t, and its average power consumption per node that is estimated at submission is multiplied by the number of nodes to get its power consumption, which is added to the current global power consumption (same for oracle but with true average power per node)
- **Ending event :** Job j ends at time t, then we update the online models and we remove the job's power consumption estimation from the current global power consumption estimation (same with oracle for the latter)

The upper plot of Figure 2.6 shows the global power estimation results over time, and the lower plot shows the relative error of the different models compared to the oracle. The online IBmodel (2.3) without weight adaptation of the jobs underestimates the global consumption given by the oracle by 5% to 10%. The errors of the weighted offline IBmodel (2.2) peak many times. This suggests that some jobs have profiles that the model did not see enough during training and that they impact the estimation randomly in high proportions. The model needs to be retrained using more recent historical data to improve its estimation, although the spikes will reappear as soon as training stops.

On the contrary, the online and weighted model (2.4) gives a much more consistent estimation, as the distribution of the relative differences with the oracle are more symmetrical with respect to 0 due to weighting adaptation. The online estimation seems to have stabilized the errors. The peaks in the error patterns may be due to bad default values for new unknown inputs, but as the model is still learning, it only sets a meaningful value for those inputs once a job has ended. Thus, the error remains low even after some time. The absolute deviation of the predictions compared to the oracle is 99% of the time under 12.7 kW, with a mean of 2.40kW. The relative deviation for 99% of the time is under 10.4% with a mean of 1.68%. The relative errors approach the measurement precision of the IMPI interface that was used to collect the data.

2.6 . Conclusion

2.6.1 . Discussion

Our work brings and solves several essential consideration for practical use of the power consumption prediction for power-aware scheduling often neglected in others :

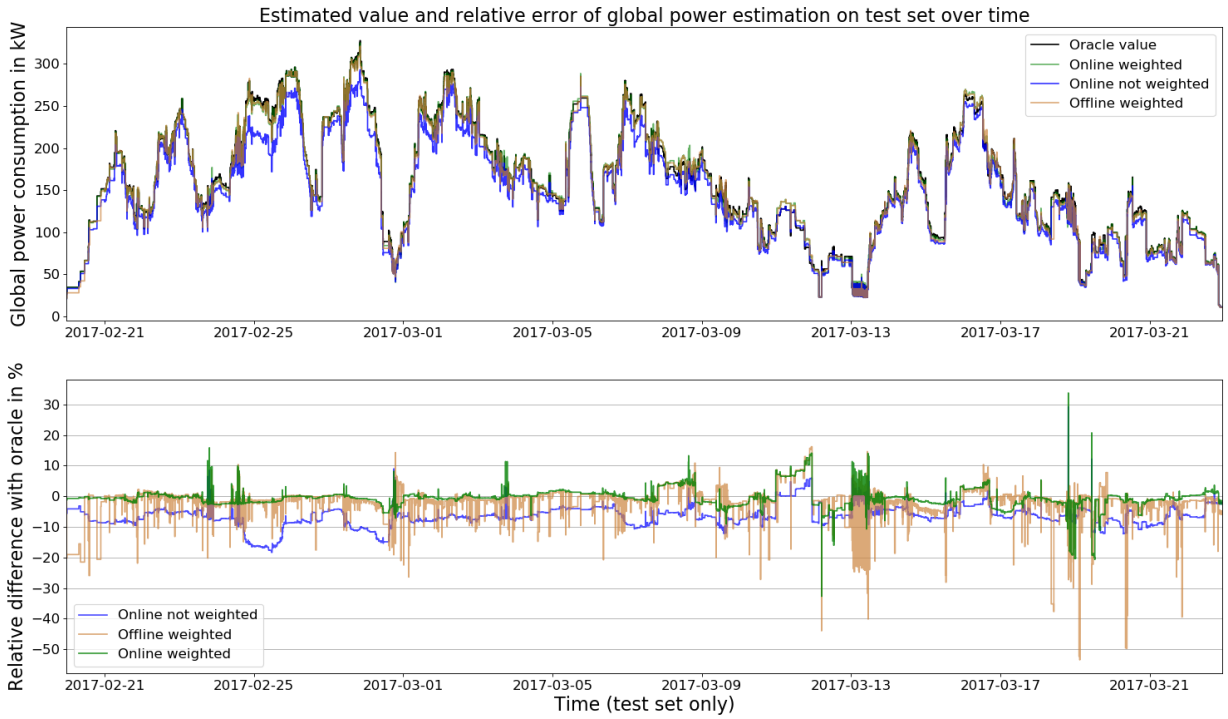


Figure 2.6 – Assuming the power consumption of a job is constant when it is running, we can compute the evolution of the global electrical consumption of the whole computing center. We call this estimation the oracle estimation, it is displayed in black in the first plot. Using the estimate of meanpow, we can also compute the estimation of global consumption that the RJMS can monitor through scheduling. We plot the estimated values and relative error with the oracle estimation.
Up : Evolution of global electrical consumption in Watts over the test period for the offline and online, weighted or not, models.
Bottom : Difference between values predicted by the three models and the oracle as percent of the oracle value

— The input data for our prediction are carefully chosen so that they are trustworthy and available without additional extraction of information in the job script of binary
— We identify a relevant prediction target : average power per node. We find by exploration that it tends to be close for many job, which was expected as it is not very dependent of their duration or number of used nodes and as many jobs are redundant once the contribution in variability of these two quantities is removed.
— We propose a model already production ready and interpretable. We know exactly how the model build its prediction. This allows to make variation of the model depending on what we want to optimize with power-aware scheduling by SLURM.
— We identify that the most expected application of our predictions, global power consumption monitoring, need to use an estimator that weights jobs contribution. None of the previous works have considerations of this fact.
— However our model performs well because jobs are very redondant which does not prevent it to overfit. No study has been made to verify that we can expect a such jobs' features distribution on other computing centers. Our study needs replication on other computing centers to assert it.

This work shows that it is possible to build an efficient model to forecast the power consumption based on the exploitation of a historical log database from the SLURM RJMS data collected from the industrial computer center COBALT that is only composed of user inputs of jobs and associated energy consumption measures. This model computes an accurate estimation of the average consumption per node of the submitted jobs using the redondancy of the information provided to SLURM by users. This instance-based model brings several advantages. It is interpretable and shows that jobs on the COBALT computing center have a power consumption that is well predicted by the GID and the number of tasks per node. We show that instance re-weighting and online computations implemented in the IBmodel are necessary to provide a prediction of the global power consumption at submission time that is not underestimated and to stabilize the relative error by avoiding the concept drift issue [Webb et al., 2016] entirely. The proposed model has a relative error that is of the order of the relative measurement error of the data, which indicates that the IBmodel's performance is already satisfactory. This model will be a good candidate for the achievement of consistent power-aware scheduling for other computing centers with similar informative inputs.

2.6.2 . A finer monitoring with time evolving data

The next step of this work is to evaluate the capability of the instances model for other computing centers with different behavior of users. This work should be extended by studying the instantaneous power consumption of jobs with time evolving data. Accounting for instantaneous power consumption will allow regulation of each job with a power cap and will enable jobs to be redistributed with more precision.

Time evolving data about power consumption were not available at the time of the study. But other time related data that were available can be used to monitor the evolution of the computation and determined the type of job that is running. This information can then be used to improve the power consumption prediction. The most characteristic time evolving data to use about a job is often the communication pattern. The chapter 4 focuses on the exploration of these.

3 - Statistical analysis and visualization of Log data

Everything that happens in a computer is a sequence of events. Each event is deterministically obtained from previous events and current state of the system. The programs instruct which events should happen at the next tick of the internal clock depending on these parameters. Those programs are written by programs themselves (a compiler is an example of program which writes another program) or humans with different purposes (the users or administrators) and programs interact with each others. This means that at the most fundamental level, monitoring a computer or a computing center is monitoring sequences of event. But they are the result of a many demands of humans that generally do not know each others. The number of events is so huge that it is already very hard for an expert to get the full implication of which events a source code could generate once compiled into a program in machine code. Given how complex this task is already, doing the contrary is practically impossible : a detailed code that can be understood by humans is impossible to deduce only from the sequences of events a program generate.

Even if the data about the use of an HPC center should take the form of sequences of all the events, this is never the case in practice. We do not need and cannot handle that many details, some filtering must be applied by humans to make it possible to grasp what the computer's behavior. However, programs are interacting with each others and not all programs are written by the same human, not even humans from the same computing center or even company. Thus, software developers are the only ones who really choose which events will be worth noticing, and they generally choose them so that they can find the origin of bugs in the software they wrote, not monitoring the whole computer.

These events are the most detailed data available for the monitoring of what is happening or happened in the computing center. But given that they are the aggregated result of the choices of many software developers, it is already very hard and time-consuming to read and understand them by a single human. Thus, it is rarely used directly to monitor the computing center but rather forensic analysis since it becomes worth digging once we are sure there were an incident. That's why it is so critical to find ways to make them easier to grasp. We try to propose visual representations of such data.

We first present which data can be used and how it should be preprocessed. Then we discuss how events can be aggregated by counting them to obtain data that is easier to process with statistical methods when it is possible. And we finally show some results of visualization and try to interpret them.

3.1 . Preprocessing of events and logs data

Not all internal events are relevant to monitor the computing center. Moreover, a simple event record by itself does not bring much information, we rather need to know how to link all recorded events with each others. This requires to identify them. We describe the different data we can

collect on such events, and discuss how they can be counted so that they are simpler to handle with statistical methods. We show how the current practices could be changed to be able to apply such methods on the most general type of recorded events.

3.1.1 . Extractable events in HPC

Event data collected can be of two very different types that we will describe. It can be a timestamp with a message indicating what is happening, or it can be the increment of a counter keeping track of the occurrence of an event or a category of events when they are too many of them to track the timestamps of each one. When the event is a message, this message's format is following conventions to store it at the relevant place and sometimes to also make it easy to use for tracking the state of system.

Unformatted syslog files

The logs are recorded events by the system which were evaluated as relevant by software developers themself used to monitor the execution of a program but also to get an overview of its state in the past. In its most generic form, it is a sequence of log lines which are strings of human-readable characters or messages along with the timestamps of record. Although the text is human-readable, it is highly dependent on how the software developer formats it and fill it with values of interest which makes the text a semi structured content.

Syslog is the most commonly used client-server protocol of messages logging. The syslog protocol defined several optional fields that are added before the message depending on how the administration of the computing center set the logging server. Although the choice of fields added to the log message is up to the administration of the HPC center, most of the HPC centers will at least add the following fields to the message :

hostname The hostname is the name of the computing node running the process which emits the message. This field is very important in distributed computing because it allows identifying if a node is acting differently from others.

binary name This is the name of the executable file that is executing and producing the message. If the name is well and uniquely chosen, this field allows quickly finding which source code may generate the event. If the event is generated by a kernel module, this value is `kernel` and the name of the module can be generally found in the message and used instead.

Process IDentifier (PID) This is a number to uniquely identify a running process at a given point in time. Concurrent programming may use several processes in parallel in the same node, this number is then the only way to differentiate if two logs are emitted by the same process or not at a given time. However, two different processes may have the same PID at two different points in time.

Depending on the choice of the administration and software developers, the message itself may follow more format conventions defined officially or not. The only consistent fields in the resulting files are the ones above imposed by the administration. No convention can be imposed on the software developers of the program running in the computing center. They tend to apply the same conventions in all the logged messages for the same software but there is no such guarantee for logs coming from distinct software.

The resulting data of syslog is then human-readable text but poorly formatted. This makes it very hard to use for automated event detection, although we expect the most information about

a failure will be written in it as highlighted by [Xu et al., 2009b].

Libraries or functions calls

It is impossible for one administration to change the source code of programs the user wants to execute. However, the source code is not enough to run the program, it requires using shared libraries that the administration installed on the computing center. The users use them to compile the code on the computing center nodes into an executable optimized to work on the nodes of the computing center and extract the highest performances.

This is where the administration can interact with code submitted on the computing center, the libraries can be tweaked to log more events related to how the libraries are used. One simple event that can be logged is the call of some functions of the library. Users may voluntarily use such tools called **profilers** to observe some pattern during the program execution.

SeLfiE is a light profiler made by [Laurent Nguyen, 2017] and available on IRENE and CURIE computing center, it tweaks the commonly used libraries like OpenMPI and returns in syslogs some aggregated count of such functions calls once an execution is done. It also collects usage and efficiency data of the node using profiling tools freely available or provided by the hardware manufacturer. The logs are formatted to be easily collected and stored together in a structured database.

SeLfiE is always used on all computations send to computing nodes in the case of IRENE and COBALT. A profiler may reduce performances of user programs. The Vampire profiler [Knüpfer et al., 2008] reports for example that each event introduce an overhead of almost 1 µs and it does not depend on the duration of the event. Thus, the overhead is very significant for frequent short events. The storage of the data also require periodic flush of memory which also introduce a significant overhead that will disturb program execution. SeLfiE only counts the calls of given functions and returns the aggregated on the fly result at the end which avoid doing memory flush so that it is light enough to not disturb performances. [Laurent Nguyen, 2017] claims that users will not notice any change during execution. The resulting data is a count of function calls and other aggregated profiling data for each process sent to the logging server and can be efficiently extracted.

Formatted Logs or Errors Codes

A software tends to use its own format convention when logging events. This is particularly the case with programs where the errors tend to be non-trivial to detect because they don't interrupt program execution either. This may happen if there is no way for the software itself to acknowledge the error (like deadlocks, when each of two threads is waiting the other to free a resource it needs to continue) or if the error does not justify stopping the execution but may have hidden consequences.

Storage management is one of the main source of errors that are hard to characterize and detect. Indeed, failures of storage hardware are considered random. The hardware itself tries to minimize such failure and may try to work around the failure, making it hard to determine if the issue is still there or not. Such errors may affect any part of storage, so it may affect the software differently depending on the type of data corrupted.

This explains why HPSS (High Performance Storage System), a very popular software used for storage management in HPC center used at the CEA, creates and details in its documentation its own message convention to facilitate forensic analysis of its logs [hpss collaboration.org, 2021]. The error codes returned in HPSS logs allow defining events and to tell if two logs describes the

same event of not. It is then possible to discard the text of the message itself and focus on the pattern of occurrence of the events.

3.1.2 . Issues with available data granularity

A set of timestamped logs is the most precise data we can get. But this is also a format difficult to use with statistical methods because there is no numbered data. To handle profiler data the same way as logs, we propose to aggregate logs by counting them. This requires to specify which logs are counted together as the occurrence of the same event, this can be interpreted as a choice of smoothing or trade-off between losing information and being able to generalize. We also noticed that log parsing seems to be required to define a good criterion to aggregate unformatted logs.

High count data definition

Count data is very common in many areas like biology, economy... They are obtained after applying the simplest yet the most generic aggregation : counting. We first define a criterion and for each possible value of this criterion we return the cardinal of the observed elements with the same criterion value. More formally count data can be defined as such :

Counted elements are taken in a set Ω. A criterion is a mapping $f : \Omega \to \mathcal{C}$ where $\mathcal{C}$ is denumerable set which define categories. Ω don't have to be a discrete or continuous set. An observation is an element of Ω^m where m is a finite integer which is the number of observed elements. We call count data a dataset which is the result of the following mapping :

$$\begin{array}{rccc} \text{counting}: & \Omega^m & \longrightarrow & \mathbb{N}^{\mathcal{C}} \\ & (o_n)_{n\in\mathbb{N},n<m} & \mapsto & (\text{card}(n \in \mathbb{N}, n < m | f(o_n) = c)_{c\in\mathcal{C}} \end{array} \tag{3.1}$$

The criterion is generally obtained by a combination of these two types :

Categories : Observed elements have a feature which is discrete, it can be a number or more generally a category (element of a finite set). The criterion is then the value of this feature. The count data obtained by counting mapping is the number of elements observed of each category

Window of values : Elements have a continuously-valued feature. The criterion is obtained from a split in covering intervals of the continuous space and is a unique identifier of the interval in which the value of the feature is found. For example, the observed elements can be events with a date. Time is split in regular windows of the same duration and the count data is a time sequence of the number of events in each time window.

Both can be combined using tuples of criteria as the criterion. The criterion acts as a linear projection preserving the sum but in a discrete set. The information carried by features not used to define the criterion is lost in the process : it is generally not possible to uniquely recover all observed events from count data. We also cannot recover all events that were observed to deduce the count data, that's why it is aggregated data.

By definition, count data takes its value in natural numbers, hence it is non-negative valued.

Aggregation trade off at or after extraction

Our generic definition of count data suggest that we could use a set of observed events from which count data is deduced instead of the aggregated count result. But several trade-offs explain why it is not possible to work with observed events directly.

The first trade-off is that full extraction may be too costly in terms of storage, bandwidth and CPU power. A detailed profiler would reduce performance of the program it is profiling because it generates a lot of data that must be sent to storage, and it consumes CPU time. On the contrary incrementing a counter when a function is called has a very low cost and the final value is sent only once the main program halts. This is the case for the light profiler SeLfiE. The counting aggregation is imposed with a step criterion, this means that it will return aggregated data on each executable launched in a job.

Another trade off is to get an estimation of the frequency of common events for the chosen criterion through counting. Better estimation for certain criterion are possible, but counting is generic enough to be applied on any type of data and to reduce the computation time. This is very useful with a categorical criterion like the error codes of the HPSS logs combined with a time window criterion to monitor the change of proportion of observed types of logs during a time window.

In the case of SeLfiE data we have no choice, we can only estimate if we do not lose too much information in the process. However, we can choose the criterion of aggregation in the case of HPSS logs, the aggregation criterion must be chosen so that the relevant information is not lost, but it cannot be too specific to avoid over-fitting.

Defining a criterion for unformatted logs

At first glance, there is no clear criterion for counting aggregation with unformatted logs. The criterion can be based on the syslogs fields that come with the message of a log line. We could count the number of log lines send during a time period by a process running on a host using the hostname, PID and timestamp fields. However, beside timestamps, all fields are optional. The other issue is that the amount of log lines carries little information about what is happening, a critical error may or may not produce many log lines and a well behaving program may or may not log many events when it is used normally.

There is however a way to aggregate log lines while preserving the semantic contained in the message. When a program sends a log line to the syslog server, it must first form a string of character which will be the message. The string may be just a way to identify which part of the source code is currently running. Such string is written in one piece in the source code and is sent to the syslog server directly if the program reach the part of the source code where it instructs to log it. This means that the same message is associated to the same event in the source code under the assumption that the software developer will not use twice the same log message for different internal events in the source code.

However, we cannot assume the other way around. The same event in the source code may result in different messages in the log files. The reason is that messages may also inform about the state of the process running or even the computer. In this case the same strings in the source code may result in different log lines once send to the logging server. Such strings have a special syntax and an extended set of features to fill part of them with other strings deduced from variables. This is called a format string. In this case, two different messages may be associated with the same event in the source code. For example, `systemd` (one of the mostly used system and service manager) can log an event using the format string `"Invalid loader entry file suffix: %s"`. The `%s` is a syntax token of the string to indicate a part that must be filled, in this particular case it will be filled with the path of a file in the storage. The path of the file may be different depending on the configuration of the computer resulting in different log line, however the event is the same in the source code.

The definition of an event through the source code is very similar to the error code used in HPSS. The unformatted log lines could be aggregated using the associated format string as a criterion. This choice of granularity preserve the semantic features contained in the message and may produce enough redundancy to learn significant pattern. The remaining issue to practically aggregate unformatted syslogs is to deduce the format string from the log message.

3.1.3 . Identification of issues with parsing

Log parsing is not a simple task but seems to be a requirement to the application of any statistical method on unformatted logs. We detailed the current state of the art in automatic log parsing, and it seems to not be reliable enough for now. We also discuss some good practices for parsing with data available today and what could be done with humans dedicated to this task.

Preprocessing before applying machine learning

Unformatted logs are the default type of data available on all computers. This makes the analysis of unformatted logs the most interesting application for HPC center monitoring. We also know that several distinct log messages can be associated with the same category of internal events when they are build from the same format string. However, this string is not available in the logs record, we must extract it. This step is called **log parsing** or just **parsing**. Using the different logs and their messages, a model can be used to solve the parsing issue. It must return the format strings written in the source code in the form of a template where the part that must be filled are replaced by wildcards (often noted "*" in log parsing literature) or a generic type of content that must be used to fill the string (that we will denote `[type]` where `type` is replaced by the type of data to fill the template). Most of the log parser proposed use statistical treatment of log lines based on the repetition of tokens in log lines.

The templates extracted from the logs of one computing center cannot always be used to parse the logs of another computing center. It is clear that if the operating systems are different, the log patterns are not matching. But this can also be the case if the versions of operating systems or running services do not match. This implies also that the same templates cannot be used on the same computing center without time limitation : a set of templates may be deprecated when a software is updated. This means that the templates generally have to be learned or deduced from the log data prior to any type of analysis.

Before learning the templates with the log parser, the IP addresses, numbers (which can be in digit or hexadecimal notation like port numbers or memory addresses), nodes and users names, file paths, email addresses, URL... are first replaced with wildcards or type token because they are always variable part of the log line. Names are known or have specific patterns of characters so they can be easy to identify with a dictionary of names or a set of regular expressions (a way to describe a pattern of characters). For example, paths are strings of directory name concatenated with slashes `/` to separate directory levels, IP addresses are either 4 digits numbers between 0 and 255 separated by dots `'.'` in version 4 either 4 hexadecimal numbers between `0000` and `ffff` separated by colons `:` in version 6. One of the main discoveries of [He et al., 2016] is that this preprocessing step increases accuracy of log parsing in practice and [Ghiasvand et al., 2016] also notice that it has the benefit to remove sensitive data from the logs.

Having a good log parser is already useful prior to any analysis of logs. It identifies when two log lines are emitted from the same internal event, this allows already to have simple visualization of the timeline of events as proposed by [Ghiasvand and Ciorba, 2017]. Each log line is a dot, the type of event is an integer on the y-axis. The x-axis is the date at which the log was recorded.

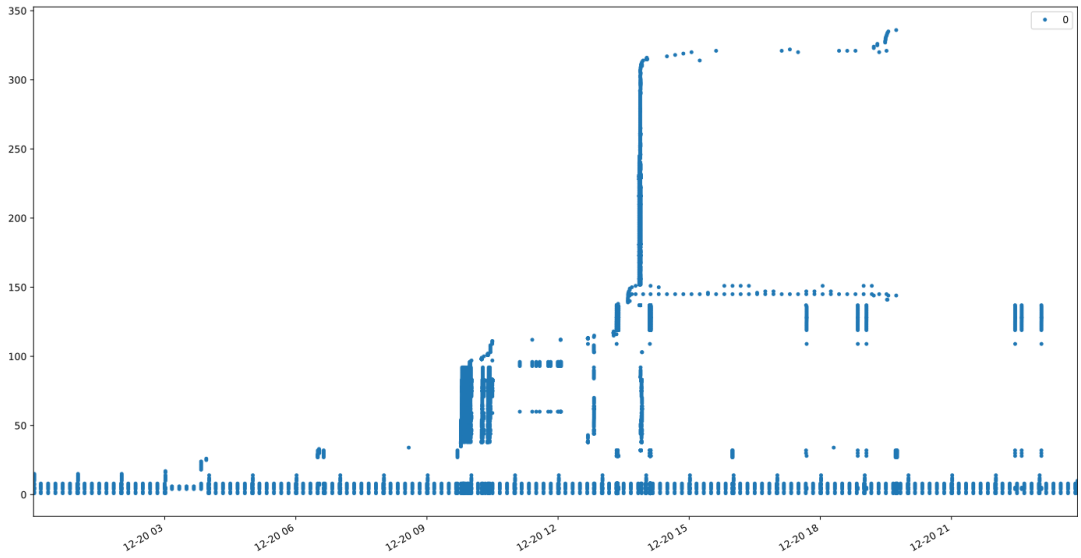


Figure 3.1 – Simple visualization of syslog events of one node using a log parser based on regular expression. The x-axis is the time coordinate over a 1-day range. The y-axis is the index of the log template ordered by the date of first occurrence.
Thanks to parsing, recurrent and usual events are easy to notice. An out-of-memory event occurred and produce a lot of rare event type, this explains the long vertical line. Zooming on this time period reveal more vertical lines. Their number corresponds to the number of processes killed by the out of memory killer to free memory.
This illustrates how log parsing already allows getting a quick overview of the log files.

On big advantage of such visualization is that we can quickly identify patterns of internal events occurring closely together in time and observe periodic and unusual events. This is much harder to do when reading the logs directly in raw text format because the human must interpret the text while looking at the timestamps : we can easily miss that more or less time pass between two log lines because they are presented one after the other without a clear visual indicator of the duration between them. An example of such visualization obtained with the set of regular expressions by [Ghiasvand et al., 2016] as a very coarse log parser is shown figure 3.1

Once textual logs are transformed into event class thanks to log parsing, a lot of models have been proposed to perform task like anomaly detection, finding root cause of problem or profiling. Such models may use methods from process mining, frequent item set mining, sequential pattern mining. However, it is hard to use them in practice on unformatted logs because such models assume perfect automatic log parsing which is not easy to evaluate and to correct.

Current state of log parsing

The main issue with log parsing is that it is hard to evaluate if a log parser is good enough for its task. Many parameters may indeed change the performance of a log parser. Most of them works well to find the templates that often appear in log records but not the rarest ones according to

[Xu et al., 2009a]. The difference in frequency distribution of templates between two logs dataset makes very hard to compare results between different log parser tested on different datasets.

The way we should measure the performance of a log parser is also unclear. [He et al., 2016] notice that it is necessary to have a good parsing accuracy for any log mining to be effective, wrong parsing increases a lot the number of false alarms in anomaly detection. They evaluated the accuracy using the F-score metric of some log parser and notice that although measured accuracy is generally high on labeled logs, log parser with same high accuracy can lead to different order of magnitude of log mining's performance. This shows that a such F-score is not a good metric to evaluate log parser.

A review by [Zhu et al., 2019] compared 13 state-of-the-art log parsers trained and tested on the same 16 datasets. They measure the runtime and quality of parsing. To evaluate the quality of returned templates, they sampled 2000 log lines and deduce manually from their expert knowledge the template of each sampled lines that is considered as the ground truth. Then they ran each log parsers on the whole dataset and compute a score called accuracy. This score is defined as the ratio of "correctly parsed log messages over the total number of log messages", where a log message is correctly parsed if "its event template corresponds to the same group of log messages as the ground truth does". This is the same accuracy definition as [Du and Li, 2016]. The group correspondence is the case where all messages from the same event according to the ground truth are clustered together. They conclude that their proposal called Drain and published in [He et al., 2017] is the best overall. However the evaluation metric favors results where the resulting templates cover many cases even if they should not be in the same group. To illustrate this, the reader may consider the case where the log parser returns one template that matches any log lines (the template contains only wildcards tokens and any number of them). Then all messages that must be in the same group according to the ground truth are indeed in the same cluster since there is only one group (all the log messages together), this implied that the score is maximal and equal to 1. It is then not surprising for Drain proposal to perform well for this metric because its default behavior is to add wildcard tokens that may never be removed. This means that a rare format string may have good chances to be matched with a more common one as having the same template : this metric favors false negative over false positive.

Another approach to log parsing is to use the source code of running programs to extract format string directly. A code is well-structured and compilers are by definition able to full parse them. This leads [Xu et al., 2009a] to build the abstract syntaxic tree (AST) using static source code analysis to extract the templates. This method has several advantages : it does not rely on statistics of the log record, provides much better guarantee that rare and even unknown events will be parsed correctly and the deduced wildcards of log pattern can even be replaced with a type of variable that can be used to build more interesting features for anomaly detection as done by [Xu et al., 2009b]. However, static source code analysis must be run on each program that appears in the log, but it may not be available, in particular when the software is set up by the user directly. Different versions of the same software may also use different format string so software differences between each version running on the computing center must also be monitored. Moreover, they also notice that some ambiguity remains because of language-specific idioms, recursive subclasses and "bad logging practices".

We also noticed that some logs may encapsulate log of others software, adding ambiguity in what should be considered as fixed part of the logs. For example, the Secure ScHell Daemon (SSHD) may log messages from a Pluggable Authentication Module (PAM). The PAM's message is than a variable that must be filled in SSHD's source code but it is build using a fixed format

string by PAM which also returns an error message from another software. One example that we expect will be common in every HPC center :

```
sshd_user[PID]: pam_echo(sshd_user:account): Cannot open [file]:
No such file or directory
```

Depending on the point of view and context, we may interpret this log lines as the message `[file]: No such file or directory` (where `[file]` must be replaced by a path to the file) coming from the file manager saying a file is missing which may indicate a critical issue with storage if we observe the same message with many files. It may also be a common message encapsulated in PAM's log `Cannot open [file_manager_log]` telling us that it can't print the contents of a file to the user currently trying to log in because this file does not exist since this login does not too, which is not a critical issue at all, the user may just have made a mistake while typing his login.

The previous log parsers aim to return templates that are associated with format strings in source code so that we can identify which logs line results from the exact same internal event. But with the recent advance in natural language processing, [Aussel et al., 2018] proposed to use semantic techniques (like stemming, synonym replacement and removing stopwords) instead of parsing. This avoids the issue of ambiguous template but it also implies that different messages that share a lot of words will have high probability to be wrongly identified as the same internal events. [Aussel et al., 2018] conclude that this is rather an advantage because log mining seems to be more accurate on the result of semantic technique. The runtime of using such techniques is not discussed.

In conclusion there is no method for log parsing for now that is reliable enough to be used without a human regularly monitoring what the parser is learning, even when the source code is available. This can be explained by the fact that logs were made by the software developers, for software developers, not to monitor a computing center. There is no consistent way to consider together logs from different software and this explains why parsing is not a well-defined problem and is very ambiguous. A part of solution must come from the way logs are produced : better log practices in order to make logs easy to parse.

Toward good practices for better automatic parsing

Preprocessings are almost always applied on logs data before using a log parser. Experts often applied rules to partially parse the logs using their knowledge of the format of certain variables. But there are also preprocessing applied that seems less justified by expertise.

The very first preprocessing that is often overlooked by domain expert is the choice of rules used to tokenize of the log lines. Tokenization is the step in language processing where the text is cut in pieces which are assumed to be the smallest semantic unit which can be compared across different lines. The tokeniser choice is often made through the choice of separators, the characters in the string of text which delimit the tokens. The most ubiquitous one is the whitespace. However, it is also useful in logs to consider the equal sign = or the colons : too as done by [Platini, 2020]. [Aussel et al., 2018] used all punctuation marks as tokens delimiters. The best choice depends on what is found in the log data. Brackets `[]`, braces `{}` or semicolon `;` are syntax elements often used to print generic and structured data called object. SelFiE returns its aggregated data using the JSON (JavaScript Object Notation) in the syslogs which use these three punctuation marks to print object fields and value for example. The Linux kernel may also use many special characters when it logs backtrace to return a nice table directly in syslog. Finally, some characters like the dash - can be really ambiguous as delimiter because it is often used in file names and so should not be considered as delimiter but it can also be used to concatenate identifiers with a

numbering format, like a software name followed by its version, for which it can be interesting to consider the dash as a delimiter. The tokenization can have a strong impact for certain log parser, in particular when the log parser assume that the number of token by log lines is constant when emitted from the same line in the source code which is the case of many log parsers (like [Platini, 2020], [He et al., 2017]). But its trade-offs are rarely discussed.

Another common practice is to replace all numbers by the same token. However, the benefit is also rarely discussed. Depending on what they represent, number generally have a typical range and some value may deserve a specific treatment. The most striking example is the numbers used for exit code which is a number that a program may return to the operating system when it ends to inform it that it ended normally or not. Thus the three following log lines emitted by SLURM each carry a very different meaning (`[number]` replaces a number in the original log lines) :

```
task [number] ([number]) exited with exit code 0.
task [number] ([number]) exited with exit code 1.
task [number] ([number]) exited with exit code 127.
```

The first log line says the task identified by a number and the PID exited normally so there is nothing to worry about. The second says it exited with errors so the user should modify its code or working environment to make its job running normally. The third says it exited either because the command was not found either because job cannot be executed. While the first and second case are nothing to worry about for the HPC center administration, the third can be much more critical : if it appears because shared libraries are not working, many users will not be able to use the HPC center as intended. [Xu et al., 2009b] actually include in their anomaly detection model the abnormal appearance of some value like this in variable value of log lines. They discriminate between actual state variable and identifier numbers based on the number of distinct values that can be found in place of the variable part (they quote in particular the POSIX norm of exit code as example of state variable). This example shows that lot of important information can be lost when replacing numbers by a simple token at parsing step and we must be more careful about what is replaced by wildcards or type tokens and how.

A partial solution to the problem of number replacement is to use tokens that also carry information about the range of numbers they are replacing. Some range of numbers are more common than other for certain identifiers or variable depending on the context. In informatics, typical ranges are often delimited by powers of 2 (so that only the highest weight bit is flipped to change the range). For example, port numbers between 0 and $1023 = 2^{10} - 1$ must be used by the system processes and not user by convention. So it may be interesting to indicate that a number is less than 2^{10} when replacing it by a token to eventually catch a different pattern if a user use a system port for example (it could be a potential attack). The previous example with exit code also shows that it could be interesting to know if an exit code is close or not to $2^7 = 128$ or not, and 0 and $1 = 2^0$ should also be treated as individual case. The same idea of keeping information about the range of number is also proposed by [Platini, 2020] where the length of the substring replaced by a token is kept so that it is easy to differentiate memory address which are always 10 characters long from other words with numbers in it.

Finally, another solution is to use a semi-supervised framework with active learning. [Carasso, 2007] proposed for the monitoring software Splunk to use mutual bootstrapping introduced by [Riloff et al., 1999] to get fields of logs and extract their values. A field is the class of variable we expect to fill the wildcards. A big advantage of having a field result is that it allows using models as [Xu et al., 2009b] based on the occurrences of a variable of a certain type. Mutual bootstrapping extract templates with fields and variables either using seed word for the field to

find, or known templates. The most common method is the Basilisk bootstrapping by [Thelen and Riloff, 2002]. A scoring method is defined for words as value of a field or for templates, using occurrence of words in a field value or template matching sentences or modern NLP model as [Hu et al., 2020] recently proposed with an AutoEncorer for example... From the words that are known to be variable of certain field, it replaces the occurrences of each word by its field type to get some template proposals and the best scoring ones are stored as known templates. From known templates, it extracts the words that are value of the field in the template and store the best scoring words result as a dictionary of known values for each field. These two steps are iterated, and it is very easy for a human to discard template by specifying bad value for a field or the templates directly. This makes it easy to build an active learner that can be improved and updated. Mutual bootstrapping is useful to extract information of text that are known to always use the same sentences patterns or with very specialized vocabulary. This is the case for logs data where sentences are well-defined by log lines and reused patterns which are the format strings so it could be an interesting way to build a database of log template. The main problem of mutual bootstrapping is collision which happens when the mutual bootstrapping result in a field value shared with another field. A conflict must be resolved with a heuristic when it happens, this can be done by the human or a heuristic, overwriting the result being the most common one proposed by [Vulić and Moens, 2013]. There mutual bootstrapping seems to offer a simple and clear way to build and maintain a database of templates and variable to monitor by the operator of the computing center.

To conclude, the best solution to log parsing would be to not have to run a log parser at all by changing the log practices. Some software developers, like the ones of HPSS, may details in their documentation the format strings they used to build their logs of events. This practice should be more common in software used in HPC centers to help monitor. However, a simpler solution in the shorter-term for generic log in syslog files would be to add as field of a log line an identifier of the format string that generate the log messages. This is analogous to the result returned by the work of [Ghiasvand et al., 2016] which return a simple hash of the string of the message after processing it with regular expressions. Adding this supplementary data was also proposed as a good direction for log parsing by [He et al., 2016] and following the proposal of [Salfner et al., 2004]. Log parsing is a key step in automatic log processing and it seems very unreliable to let a model perform it with the currently available syslog data without expert knowledge in the loop. We think this is the most critical limit to the use of any log mining model to monitor an industrial computing center and adopting a standard to return an event identifier with the log message should be the top priority of any operator who wants to use machine learning on syslog data for critical monitoring in production.

3.2 . Aggregation into high count data

There is no straightforward way to visualize the content of logs, this is only possible on well formatted and known ones. That's why we focus only on the occurrences of specified events or logs that are well formatted for the rest of this work. We need to aggregate some events together to extract information about their occurrence. The most simple way to do it is to count events, as it is already done for functions calls event at extraction.

We first present and propose a preprocessing of the data extracted by SeLfiE and logs from HPSS and describe the different trade-off when aggregating event logs. We observe from the

comparison of the two resulting datasets that our visualization must be scale invariant, so we decide to apply a logarithmic transformation to enforce this constraint, which is not an easy task.

3.2.1 . Exploration of the features of aggregated data

We first present HPSS logs data and discuss the different ways of aggregating them into count data with their trade-off. Then we present the count data extracted by the SeLfiE profiler running at the CEA and describe a preprocessing needed to aggregate them by jobs correctly. Finally, we compare the two datasets and notice that by construction, two samples that can be considered close can have different scale because of the aggregation.

Counting aggregation criterion for log data

The data returned by SeLfiE is already aggregated into count data for practical reason at extraction. The events set from which SeLfiE data are extracted is however very similar to the HPSS's event described in its formatted logs. We propose to create an unsupervised visualization tool to help either with the monitoring of jobs with SeLfiE data either with the storage system with HPSS's formatted log. To allow the same visualization on SeLfiE data and on HPSS's logs data, we must apply a counting aggregation on the HPSS's log data to work with count data only.

While the SeLfiE data are mostly function call events already aggregated based on the step criterion, we have to chose how we aggregate the events from HPSS's logs to also get count data. A recorded event logged by HPSS is a string of the following form :

```
Record type=ALARM, Event time=2020/03/07 08:27:56 CEST, Severity=MINOR
    Subsystem=CORE, Message#=3073, Error code=-1436
    Desc name=Core Server, Routine=ss_WriteDisk:Mover 0 (line 708)
    PID=22735, Node=node1337, User=
    Type=SOFTWARE_ERROR, Object Class=40, Request Id=49975236
Active side of copy operation failed: Resource locked
```

The first block is a header containing meta-information on the message that comes after. There are many fields with different range of values :

Record type : This value is the general class of the message and is categorical. It is `ALARM` in the example, indicating something that requires administrator attention. It can be `DEBUG` if it could be useful for troubleshooting or `EVENT` if it is an informative event for the operator even if not worrying (a starting or exiting event for example). Other value are possible like `TRACE`if the logger is set to details as much as possible what HPSS is doing but we did not see them in our data.

Event time : Timestamps of the message, it is a continuous variable discretized to the nearest second.

Severity : The impact that the logged event can have on the running application or system. It can be empty if nothing will happen (that is the case of all `EVENT` type log messages), `WARNING` if it is abnormal behavior but automatically handled, `MINOR` if an issue must be resolved by the administration operators but the application is still functional, `MAJOR` if the behavior of the application is very different to what we might expect from it. Finally, `CRITICAL` indicates that the application is unusable or that it is broken or breaking the system. This level of severity is set by the software developer of HPSS. (Two other values are possible, `Indeterminate` and `Cleared` but was not encounter in our data).

Subsystem : Acronym of the type of subsystem or server also shown in `Desc name` which is emitting the message. It is `CORE` or the Core Server of the storage system in the example, but it can also be a mover `MOVR` (on-board system for disk loading and management like robots for magnetic band storage).

Message# : Number associated to the message. Prefixed with the value in `Subsystem` field, it forms a message identifier that can be used to look up for details in the error manual.

Error code : The type of problem underlying the message.

Routine : The name of the function that was executing when the message was logged. The name of the routine may begin with an identifier of the subsystem or type of hardware the routine interact with. `Object class` is an identifier of the class abstraction that implements this routine in the source code. The line number in the source code of the routing at which the software records the log is also given in parentheses.

Type : The class of the event recorded. It can be a software or hardware error, a service degradation, informational text...

Request ID : Identifier generated by the process logging the message. This is very similar to the process ID (PID) except a process may generate several requests. The main one is always 0 and run continuously. Like the PID, if the request IDs of two messages are the same and if they are close in time there is a high chance the two logs are recording event from the same request.

PID, Node, User : The PID, the hostname of the node and the username running HPSS when emitting the log message

The second part is the body of the log and contains the message. The first line is the message template identified by the message ID filled with value. More information can be added in the next lines. This part is not formatted and so present the same requirement of parsing to define a criterion. To define a meaningful aggregation criterion, we can however use the log header fields' value since they are formatted.

Using message ID (value of `Message#` field prefixed with value of `Subsystem` field) we can produce the same kind of timeline as figure 3.1 to get an overview about when something happened during the record without having to manually read the logs. The messageIDs are sorted in frequency order (the most frequent message have a low rank) and the occurrence of a log message is displayed as a dot with coordinates corresponding to its date and frequency rank of its messageID. In an attempt to identify relevant event, the dot is colored depending on the severity of the message. The result is shown figure 3.2.

If we were given enough relevant failures to identify, we could use the sequence of message ID to build a model to maybe forecast them. We do not have such label, so we try to provide unsupervised visualization that rectify some drawback of the one provided in figure 3.2 which could also be used for data like SeLfiE. The main issue is that all events are displayed in the same way but some pattern may be important, so we must find a way to represent pattern of events, not just event. To do that we must aggregate events together, this is done by converting event data into count data.

Our criterion is a tuple of two criteria. The first is the time periods in which the events happen. The time periods are regular windows of the same duration in which the event time is. We just have to choose a starting point and the duration for this criterion, given that the time duration of a window must be low compared to the duration of the whole recorded dataset, the starting

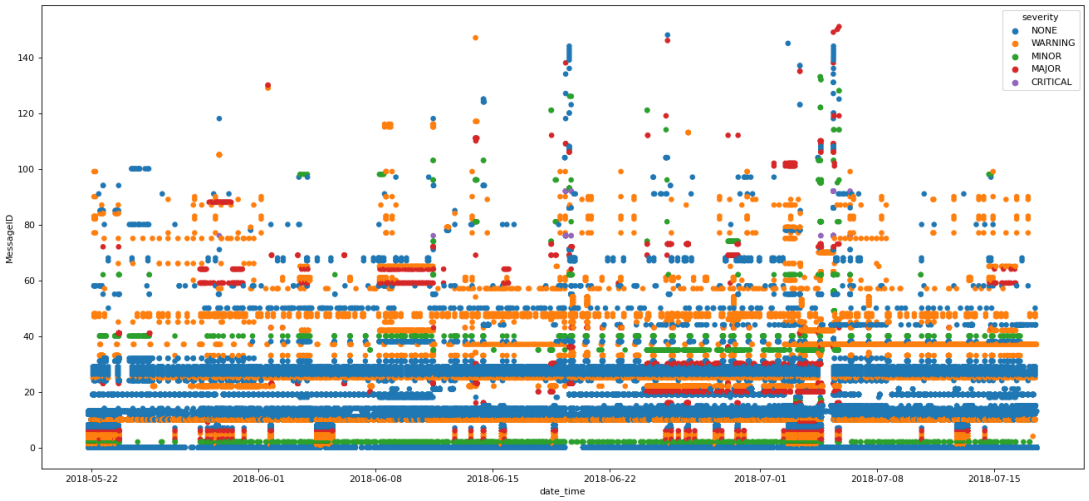


Figure 3.2 – Same visualization as 3.1 but using the available messageID to define the type of event. Dots are colored by log severity. The x-axis is the time over a range of 2 mouths. The y-axis is the index of messageID ordered by frequency. We are not aware of any reported event that deserves attention.

point does not have a strong impact so we choose it arbitrarily. However, the higher the duration of a window the smoother in time the resulting time series will be. This is because the counting aggregation act as a regular sampling combined with a low pass filter.

The second criterion is the value of a field other than event time. It can be notice that some fields form a hierarchical structure of information. A given `Routine` value implies a unique value for `Subsystem` because only this subsystem may run this routine for example. The same is true for the messageID (`Message#` value prefixed with `Subsystem` value), it implies a unique value for `Routine` because a given message can only be generated by a certain routine. This means we can choose to have a fine or coarse criterion by using certain fields and not other. A criterion is finer the more unique value it can take because there will be less event counted together. A very fine criterion would be using the messageID while the subsystem value or Record type would be coarser because those fields act as groups identifier of related messageID value.

The resulting data from counting aggregation is a time series for each unique value of the second criterion of the number of time an event with this value for the criterion was observed during the time period considered. This let us two granularity parameters to choose when aggregating HPSS log data. To choose these parameters, we propose to compare the resulting count data with SeLfiE count data.

SeLfiE count data exploration

SeLfiE data are already aggregated for each process of a job. So we don't have to choose the criterion. However, SeLfiE data are not time series, they are more job series indexed by processes. The data returned by SelFiE are logs with a message of the following form :

```
{ "selfie_version": "1.1", "utime": 85173.62, "stime": 1119.44, "maxmem": 1.09,
"posixio_time": 16.86, "posixio_count": 36148, "papi_ipc": 0.93,
```

```
"papi_vec": 0.00, "papi_mem_bw": 0.00, "mpi_time": 10603.25,
"mpi_count": 784714, "mpi_version": 3.10, "mpi_libversion": "open mpi",
"mpiio_time": 0.00, "mpiio_count": 0, "mpiio_version": 3.10,
"mpiio_libversion": "open mpi", "USER": "saillant", "SLURM_JOBID": "276978",
"SLURM_STEPID": "0", "SLURM_PROCID": "1664", "OMP_NUM_THREADS": "1",
"wtime": 86383.90,
"command": "/opt/abinit-8.8.2/Atos_7__x86_64/intel--17.0.4.196__openmpi--2.0.2
/default/bin/abinit" }
```


Like the header of HPSS logs, SeLfiE returns formatted logs with fields and values, it uses the display convention JSON for object. The field gives information over several independent parts of the usage of resources by the job.

CPUs' time : `wtime` is the wall time ("wall" for "wall-clock"), this is the elapsed real time of execution of the program. `utime` is user-time or the sum of the duration the CPUs are used in user space, meaning the period when they execute instructions which are not from the kernel of the operating system. `stime` is as `utime` but it is the system-time for kernel instruction. `wtime` is not always the sum of `utime` and `stime`, this is only true for jobs using one unique CPU. For example `utime` can be higher than `wtime` if several CPUs are used by the process. Otherwise, the sum of `utime` and `stime` is approximately the wall time multiplied by the number of CPUs.

RAM maximum usage : `maxmem` value is the maximum resident set size or the maximum amount of RAM in GigaBytes allocated by a process. Given that the nodes of the computing center observed have at most 192 GigaBytes of RAM, this value must be less than 192. If a process needs more memory while the RAM is full, some processes will be killed or an auxiliary memory in storage will be used but this value does not count it.

Standard Inputs/Outputs : the usage dedicated to reading and writing data in the disk storage is observed through the count of calls of standard functions to interact with data in storage returned as `posixio_count`, the time spent executing such function is returned in `posixio_time`.

PAPI data : PAPI by [Terpstra et al., 2010] is the Performance Application Programming Interface and provides a consistent interface for hardware performances counters. SeLfiE collect data from PAPI and return them too. The data returned are the average number of instructions executed by CPU cycle `papi_ipc` which indicates an efficient use of CPUs, the ratio of vectorized operations (optimized concurrent operation on vectors) in `papi_vec` and the memory bandwidth (rate at which data is read or write on the RAM) in `papi_mem_bw` in GigaBytes.

MPI library calls : SeLfiE detects if the processes parallel programming library MPI [Gabriel et al., 2004] is used and which implementation and version returned in `mpi_libversion` and `mpi_version`, it tweaks the MPI library to count the number of function calls and returns it in `mpi_count`. The sum of durations to execute MPI functions is returned in `mpi_time`.

MPI I/O library calls : SeLfiE does the same with the I/O module of MPI to read and write concurrently in storage disks. The data are returned into the same format as MPI but prefixed by `mpiio`

SLURM metadata : SeLfiE also includes SLURM data in its return (the username `USER`, job identifier in `SLURM_JOBID`, step identifier in `SLURM_STEPID` and relative MPI rank of the process in `SLURM_PROCID`)

OpenMP data : SeLfiE also collects data from the threads parallel programming library OpenMP [Dagum and Menon, 1998]. In the version of SeLfiE used to collect our data, it only returns the number of OpenMP threads used by the process in `OMP_NUM_THREADS`.

Command path : The full path of the running program is returned in `'command'` field. The name of the executed binary file is the string of character after the last slash `'/'`. This data is malleable by the user and must be used with caution.

A first issue with SeLfiE data is that SeLfiE returns a raw of value for each process even if they are part of the same step of the same job. Two processes that are part of the same step are two processes working together so the data returned by SeLfiE are strongly dependent from one-another, although not redundant. This is illustrated in figure 3.3. Each scatter plot displays the number of MPI functions call against the wall time of each process. But the second scatter plot only consider the main and mandatory process 0 of each step (we draw a dot only if `SLURM_PROCID` is 0). As expected, there are fewer points in the second plot but the overall distribution of points is preserved. However, we lose information on the distribution of values of the processes of a step so we need to aggregate them instead of just sampling the first process.

But there is a second issue with SeLfiE data that must be resolved before aggregation. The values returned are either extensive either intensive, meaning values are respectively proportional or not to the "size" of the process. It is clear that CPUs' times, MPI library calls and MPI I/O library calls are extensive values and PAPI data are intensive since the latter are ratios of extensive values. It is less obvious to tell if the values related to standard inputs and outputs `posixio_count` and `posixio_time` and the maximum RAM usage `maxmem` are extensive or intensive. One way to find out is to plot them against another extensive value that play the role of reference of the "size" of the process. The real duration of a job and so the duration of a process is the criterion that all HPC center users try to minimize. The users will often run the same job with almost the exact same setup twice but with different duration : one "small" run to check that the code is computing what expected and then "large" production run where they let the program completing the full computation. This makes the wall time `wtime` the most intuitive value to represent the "size" of a job from users point of view so we choose it as reference. However, notice it may not be appropriate as reference for all values, in particular when a process also uses several threads. Indeed, we might expect the CPUs usage to be proportional to the number of used thread too if there are several of them, this means the reference could be the wall time multiplied by the number of threads given by `OMP_NUM_THREADS` (provided the threads are managed by OpenMP) or the `utime`. Figure 3.4 shows the plot in logarithmic scale of `posixio_count`, `posixio_time` and `maxmem` against the wall time `wtime` for extracted data from processes 0. If a value is intensive, the value is independent of `wtime` and we should expect no trend. If a value is extensive, then it should be proportional to `wtime` for all jobs so an increasing trend with slope 1 should be noticed in the plot. A line with slope 1 in black shows the expected direction of the trend for extensive values (the trend must be parallel to the black line if the value is extensive because of the logarithmic scale). We know that `posixio_time` cannot be greater than `wtime`, this bound is plotted in red.

The plot of the maximum RAM usage clearly shows that it is an intensive value. This is because a process can only use as much memory as the node on which it is running has RAM. Total memory usage could be extensive, we expect bigger jobs to use more memory. But the fact that we

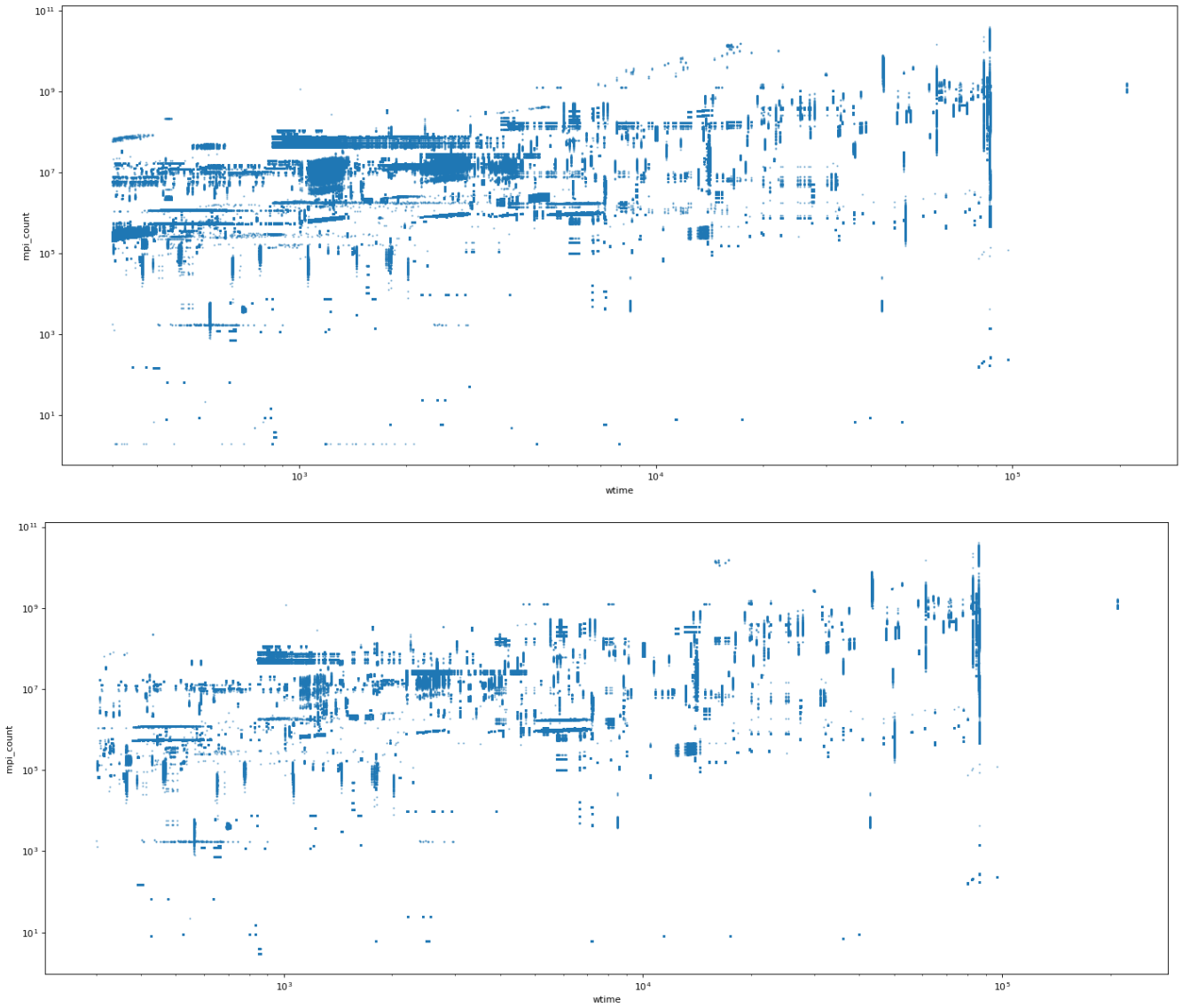


Figure 3.3 – Scatter plots of the number of MPI functions call against wall time of processes. The first plot consider all the processes while the plot below only consider the main process 0 of each step. The trend of both is similar. However, we can see that some cluster with different shapes (horizontal or inclined rectangle) disappear. This shows redundancy of processes of the same step and the need to reserve information about this shape when aggregating data by step

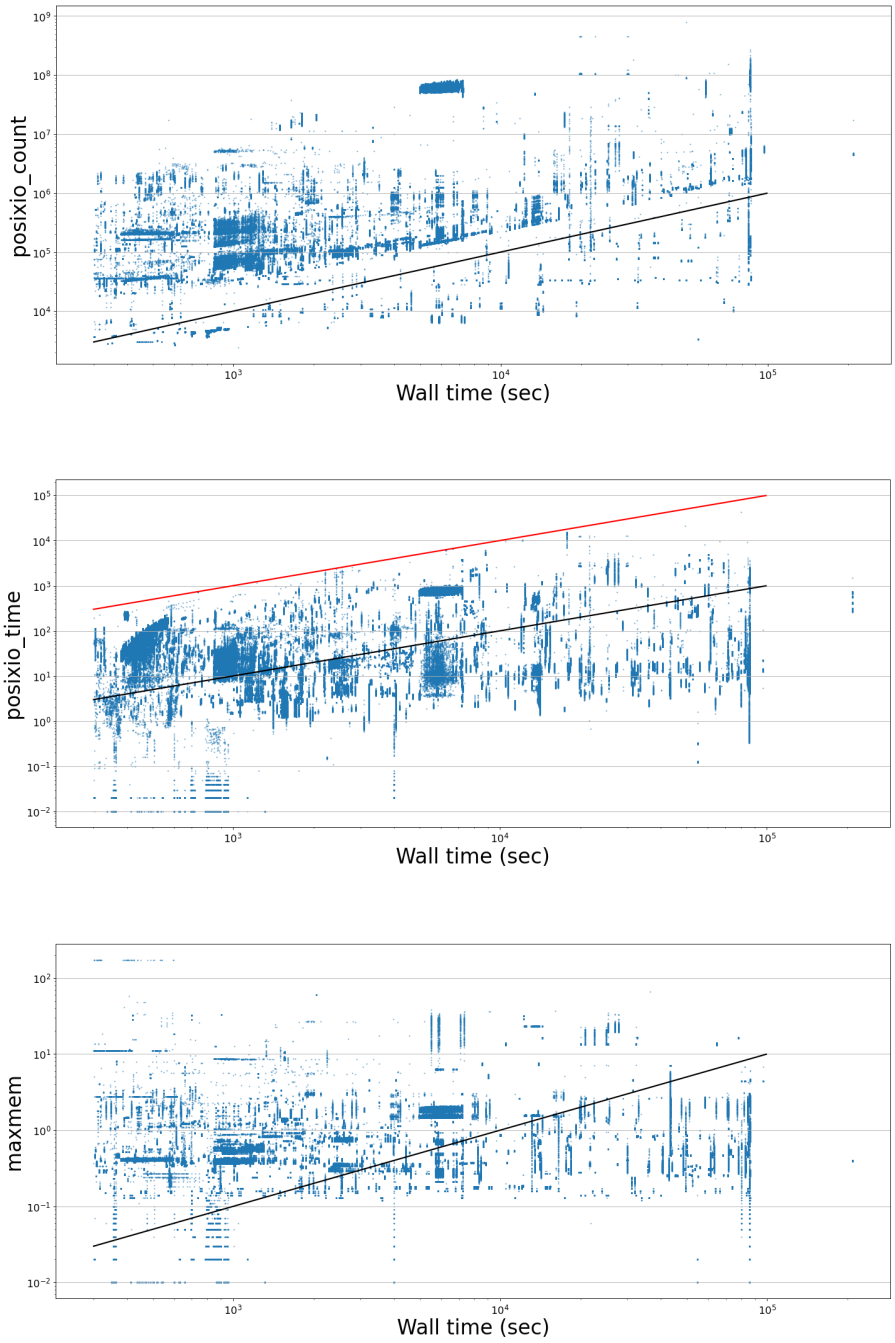


Figure 3.4 – A visual test of extensive value test for Inputs/Outputs counts, time and maximum RAM usage respectively. The scales are both logarithmic. An intensive value has no observable trend compared to wall time. An extensive value should be proportional to wall time, it should have a trend with a slope of 1, which is displayed as a black line for better readiness. By definition, Inputs/Outputs time cannot be greater than the wall time, this bound is plotted in red in the middle plot. Inputs/Outputs counts seem to be extensive. The Inputs/Outputs time seems to be an intensive value.

Intensive values	Extensive values
posixio_time	posixio_count
maxmem	mpi_count
OMP_NUM_THREADS	mpi_time
papi_mem_bw	mpiio_count
papi_ipc	mpiio_time
papi_vec	utime
	stime
	wtime

Table 3.1 – Summary of our conclusions about extensive and intensive value returned by SeLfiE on processes. Extensive feature must be divided by wall time before being aggregated by jobs.

are measuring the memory usage of process and not job makes it intensive : `maxmem` measures a memory usage per nodes, which makes this data intensive because the amount of RAM and memory on a node is the same for every one of them.

The plot of the POSIX Inputs/Outputs value is more surprising. We may expect that if the number of call of functions is extensive as shown in the first plot of `posixio_count` then the total time spent executing them would be extensive too. But this is not so obvious on the plot of `posixio_time` where it seems there is no significant increasing trend other than the bound given by `wtime`. This means `posixio_time` rather behaves as an intensive value, particularly when the wall time is high. The explanation may be that big jobs will often return intermediary results regularly so that the number of I/O calls is indeed proportional to the length of the job but the longest interaction with storage is rather initializing the domain of computation using data in storage (importing the meshes in memory for example) so that saving intermediary results is negligible. This also explain why `posixio_time` is close to `wtime` mostly when the latter is low. We choose to consider that `posixio_time` is intensive in practice.

We treat the number of OpenMP threads given by `OMP_NUM_THREADS` as intensive for the same reason as `maxmem`, a higher number of threads in a node does not imply the job is bigger because the usage intensity is bounded by the number of cores in a node. This makes `OMP_NUM_THREADS` an intensive metric bounded by node specifications as `maxmem`. However, like memory usage, the total numbers of used threads (summed over all processes of a step) can be considered as an extensive value. We summarize which values are intensive and extensive in table 3.1.

We removed from the data the processes with identifier higher than 0 to remove the dependency between samples from the same step. By doing so, we lost information of the distribution of values over processes of a step. This information can be important to characteristic different type of programs. For example, the longest process is often the one which defines full computation's duration because others need to wait for it. On the contrary, the fastest could have been more used to balance the resource usage over processes. Some kind of jobs are more or less balanced between processes. To keep such information, we must aggregate the values for each process into values for each step. One issue is that processes may have different size but we know how to transform the data to make different processes comparable thanks to the classification of fields value in intensive and extensive value. The solution is to divide extensive values by the reference `wtime` to obtain intensive ones. Then we can aggregate all value together (and we can remove `wtime` since it is the reference size). We choose to keep the minimum, the maximum, the mean and the standard deviation of each value of processes of a step for each step of our

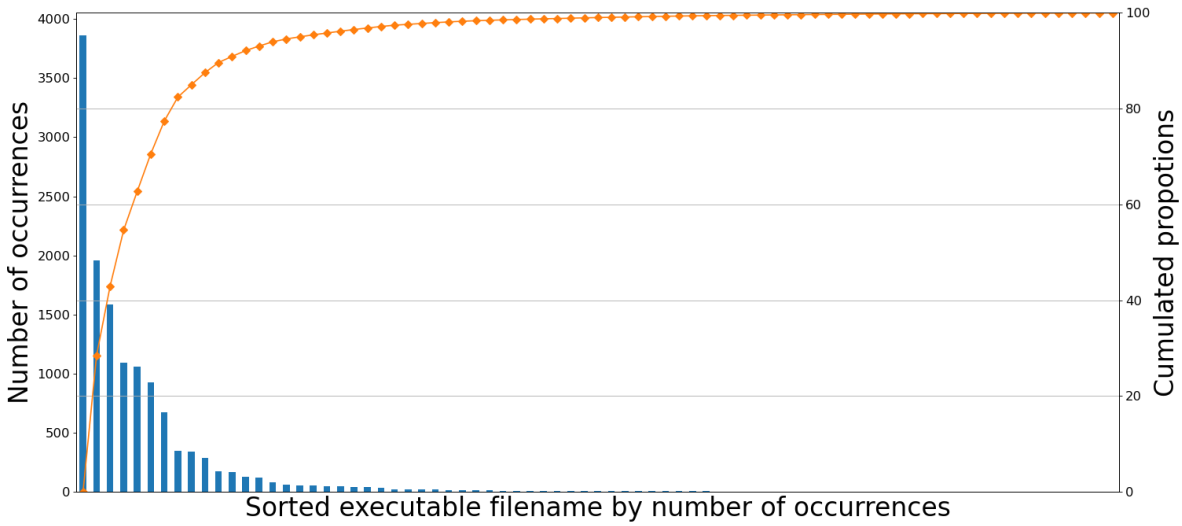


Figure 3.5 – Pareto plot of the executable filename of steps. The blue bars show the number of occurrences of a given executable filename in our dataset (scale is given at the left of the plot) and the yellow curve display the cumulated proportions of jobs in our dataset with more frequents executable filename then the one on x-axis (scale at the right of the plot). One might expect that a representative dataset would have balanced classes : the blue bars would have roughly the same height and the cumulated proportion would be a straight line. This plot shows that our dataset is very unbalanced in terms of filename (the 10 most frequent filenames represent 90% of our dataset) which can introduce bias in our representation. We resample (with replacement) in it so that each filename have the same number of samples as others.

dataset. We also add the number of processes of each step to replace `SLURM_PROCID`. The result is a dataset of step for which we have the 4 previous estimators for each value returned by SefiE taken over all the processes of the step and its number of processes.

Finally, SeLfiE data are also sampled Depending on the usage of the computing center and the number of step per job. This means that some programs are oversampled because they represent the main usage of the computing or because they appear in jobs with many steps while others are down-sampled because they are rarely used or only for few applications. This fact is illustrated in the Pareto plot in figure 3.5. Our goal is to build a visualization of SeLfiE data that allow to see if we can classify the running programs. To resolve the imbalanced sampling issue, we extract the binary name from the `command` field of SeLfiE data by only keeping the filename after the last slash `/`. Then we sample 50 times with repetition a step with a given filename for each unique executable filename value to get a new dataset in which all executables have the same number of occurrences.

Comparison between the two aggregated data

HPSS and SeLfiE datasets are both resulting from a counting aggregation. But there are still many differences between them.

First the samples have a different type of dependency in the two dataset. HPSS samples are points of a time series so they are not independent but the dependency is known. In the case of

SeLfiE data, our aggregation of processes data results in a sample of estimators of the number of events counted in each processes for each step. The dependency between samples is then given by their membership or not of the same job or step. In both cases the dependency is known and can be used to evaluate that the proposed visualization renders it clearly.

The major difference between HPSS and SelFiE data is how our choice of counting aggregation introduces the dependency between samples. HPSS events have been aggregated uniformly through time with the choice of a time window duration of aggregation. Every value is extensive, the higher the time window duration, the bigger the values. The windows'duration is known so we can easily create intensive values from count data just by dividing them by the window duration. The choice of time windows duration is then similar to a choice of smoothing with a low pass filter if we consider the resulting count series divided by the window duration. However, the HPSS events are also aggregated by the value of one of the fields of HPSS logs. This choice introduces an arbitrary aggregation of events that may be non-uniform. For example, some category of events may appear several times for one other category of event. This means that there is still unknown proportional relationship between counts of events of different categories depending on what is happening in the considered time window.

This is even more the case for SeLfiE count samples. Events are not aggregated uniformly by the SeLfiE profiler because not all steps have the same size. As already discussed in previous section, the non-uniformity of aggregation produce extensive values that are not so trivial to convert into intensive value. Even after dividing by the wall time the values we identified as extensive, other hidden features like the size of the domain of computation, the number of nodes used by the job, the load balancing between CPUs, the number of processes or thread used and amount of communications required can make a job step a "bigger" aggregation than another. Once again, this implies there is still unknown proportional relationship between the value returned by SeLfiE even after removing the wall time contribution to the scale.

The counting aggregation is a multiplicative mechanism, it imposes its scale to the resulting data. This makes the absolute values not meaningful, even if we try to remove a global scale effect. However, the origin of this non-uniformity is a macro-event of a larger scale than our choice of aggregation, this means that in both cases of HPSS and SeLfiE we should visualize the relative value of the count data, ignoring the scale introduced by the aggregation. Our goal is to build a scale free visualization for SeLfiE and HPSS count data.

3.2.2 . Scale free result transformation

The counting aggregation introduces an artificial scaling effect. The information is only contained in the relative values. This is the core assumption of a research field in statistics called Compositional Data analysis. We review how it can be applied to our data.

We first present what are Compositional Data and the main challenge to apply Compositional Data analysis to our data : applying the logarithm on structural zeros. We review how this problem is solved in the literature. Then we propose a way to deal with them for visualization purpose using the framework of missing values to give a value for the logarithm.

Compositional Data and logarithm transformation

The data carrying relative information is called compositional data. Compositional data often appear in measurements of proportions or probabilities. Composition can also take the form of counts. The field of Compositional Data analysis (CoDa) was introduced by [Aitchison, 1982]. In the most general form, the data points are real vector with positive components. Data must be

normalized to remove the global scale. Once normalized it is assumed that the samples lives in a simplex of positive data with a fixed sum. This simplex is given a new geometry called "Aitchison geometry". This geometry is the one obtained from classical vector space for which each component of the vectors is replaced by their exponential and normalized. The implicit assumption is that in a composition, a change or error is not additive but multiplicative.

The main point of this geometry is to take the logarithm of all coordinates of normalized compositional data before using classical statistical model. The data can be normalized by dividing all coordinates by one of them or any homogeneous statistics like the sum of coordinate or their geometric mean. The resulting collection of transformation is called log-ratio transformation.

Even after careful normalization to get intensive values from the extensive ones, we need to plot SeLfiE and HPSS log counts data with a logarithmic scale. This suggests that Aitchison geometry is relevant to analyze the data so they can be considered as compositional data. The number of times an event appeared would represent its proportion in the time frame of aggregation (regular time window or step running period) where all events were counted once normalized.

The main issue with compositional data analysis is by far the zero-valued coordinates. This is clear from the fact that any compositional data analysis will use a log-ratio transformation which cannot be valued for zero-valued coordinates because of the undefined logarithm at 0. This motivates [O'Hara and Kotze, 2010] to claim that log-transforming count data is a bad practice and recommend to use Poisson or negative binomial models. However, they also noticed that this practice is not so bad when counts are high.

The treatment of zero-valued coordinates must be done depending on the reason why they are present. The use-case developed by [O'Hara and Kotze, 2010] is when counts are low, which correspond to a case where an element of the composition is undetected because of a detection limit of a sensor. Another explanation of zero-value coordinate in count compositional data is a rounding error.

By construction, SeLfiE and HPSS contains zero-valued counts. However they are different from previous types. The reason they appear is because they are "true zeros". For example, the coordinate associated to the value of the field `mpiio_count` is often 0 when a process did not use the MPI I/O library at all. In the case of HPSS logs, many messageID will not appear during a time window which will also produce a zero-valued coordinate for each messageID if we choose to aggregate the events with the messageID as criterion. We call such zero structural zeros. There is no general way to handle structural zeros of compositional data and we must investigate with knowledge on the data why such structural zeros appears.

Logarithmic transformation with zero-valued coordinates

A very common practice to handle zeros when applying a logarithmic transformation is to use smooth surrogate functions for the logarithm that are well-defined at 0 and approximate the logarithm for positive values. The most common practice is to add a small constant to all values before applying the log. A common choice of this constant is 1 so that $\log(x+1)$ is 0 when x is zero. When used in a supervised learning problem, it is also possible to learn the best value of this constant as proposed by [Le and Cuturi, 2013] to learn generalized Aitchison embedding of compositional data like histogram. [Bellego et al., 2021] pointed that "many practitioners use this solution without even mentioning it because they think that adding a very small constant is not going to be harmful" while it is clear that the logarithm will expand low-values and generate a bias Depending on the range of order of magnitude of the data. They also showed that choosing the smallest possible constant is not a better choice. An intuitive explanation is that while the choice

of a very small constant makes the approximation of log very good for low positive counts, it introduces a very large distance between zeros-valued data and positive data so that almost all the variance is explained by being zeros-valued or not, no matter the positive value. This makes it hard to detect relevant direction for the variance of log-transformed positive data.

Another old practice introduced by [Johnson, 1949] is to use the inverse hyperbolic sine transformation as surrogate of log transformation which also offers the possibility of handling negative values if they have a meaning (e.g. for data that are difference of counts). This was developed by [Burbidge et al., 1988] along with an extension of the other very common alternative, the Box-Cox transformation, which uses power-based transformation. Those transformations approximate well the logarithm for high values but not low value and present the same kind of issue as adding a constant : a scale of value from which the logarithm is well approximated by the inverse hyperbolic sine must be chosen and choosing it the lowest possible is equivalent to choosing an arbitrary value of 0 for zeros-valued data which may introduce bias.

The recommended practice by critics of the logarithm transformation is to use generalized linear model (GLM) with a Poisson distribution for counts or zeros-inflated-model that will add a dirac at zero in estimated distributions, meaning that the random variable being modelled has a probability of being zero or is drawn relative to the Poisson distribution with parameters estimated with a linear model. Such model made the core assumption that the counted events are emitted by an approximately stationary count process at the scale of aggregation, which could be true for HPSS data if aggregation time period are short enough. However, we should not expect it for SeLfiE data since the data are at least aggregated over the whole duration of the processes.

In conclusion, it was already noticed that there is no generic way to handle zeros in count data as the best treatment depend on the reason why such zeros appear. In the case of HPSS data, the recommended practice to use Poisson based model could be relevant. However no recommendation among the literature seems to be satisfying for highly aggregated count data like SeLfiE. It seems high count data is different from classical count data in the sense that the zero is rather a categorical feature than a count : it indicates a class ownership that result in no data for a count feature. This makes it hard to build a common projection space to visualize the distribution of such data without creating artifacts that could lead to wrong interpretation of the shape of the data.

Mean imputation of logarithms for SeLfiE data

The previous methods to handle zeros-coordinate implicitly assume that zeros are associated with lower values of the data they represent than positive data, they are value that are lower than detection limit or rounding zeros. The logarithm transformation is handled by setting a low positive value in place of the zero which can be computed from the detection limit or an imputation model that uses the positive data to set an appropriated value for the zero of logarithm.

The order between zeros and positive values may also be ignored. Zero values can be associated to missing values. That is why a more radical solution is to remove the samples or to aggregate features with zeros-values with each others to remove them. But even this solution is not ideal if used : it requires more data for the same quality of estimation and it introduces bias in the data if the zeros are not uniformly distributed and independent of the data. Removing data with missing value does not bias the result only under the assumption that they are "Missing Completely At Random" (MCAR assumption) which means that "missingness" state is independent of the values, missing or not. Otherwise, the curated data is biased toward values that introduce less missing values.

When the data is "Missing At Random" the non-missing values are the only dependency of the "missingness" with the true data, this could allow to weight the data once samples with missing data are removed to remove the bias. This assumption cannot be verified statistically according to [Little and Rubin, 2019] and we must rely on our knowledge of the process that introduces missing values. In the case of our high count data, the zeros values are associated with the fact that a library is not used for SeLfiE data or the non-occurrence of a log event. If interpreted as missing values, the "missingness" is not random but strongly related to what a sample represents. It is clear in the case of SeLfiE data that the "missingness" of certain value are explained by the missing values themselves. This means we have to assume that the data is "Missing Not At Random", which is the worst case and imply that it is impossible to remove samples with missing value without expert knowledge of how this will bias the data, knowledge we do not have.

When it is not possible to delete missing value through samples, we must choose an imputation method to replace missing values with some values that are of the same type as non-missing values. The most common and the simplest imputation of missing values is to replace them by the mean of non-missing values. This choice minimizes the sum of squared error introduced by the imputation and so minimize the variance on the resulting data. It is strongly criticized by [Donders et al., 2006] because it introduces bias in linear regression coefficients toward "0". An interpretation of this fact is that the same value for all missing values of a feature is a constant function for which the regression coefficient is 0 mixed with the available data.

Another common method of imputation is to build a binary matrix that indicate if a feature value is missing in a sample and impute the missing value with an arbitrary value like the mean, 0 or an estimate computed from the k nearest neighbors according to a distance computed on non-missing feature of the sample. This was proposed in the case of zero handling of Compositional Data by [Hron et al., 2010] first and used by [Templ et al., 2017] to perform outliers detection using a score based on a Mahalanobis distance computed on imputed data and "missingness" binary matrix with good results. The use of an indicator matrix as predictor of a regression is however strongly criticized by [Donders et al., 2006] too. This illustrates the often overlooked fact that the best choice of an imputation method of missing values is not just an assumption on the distribution of true values of the missing value, it strongly depends on what we try to achieve. To get a satisfying solution, we need to model what is missing but the best imputation method also depend on what we are trying to achieve.

It is clear that the different jobs can be differentiated by the different libraries they use and this is easily detected by the presence of zeros on some coordinates. However, since we also want to know if other patterns are present in the non-zero data, we need to choose a replacement value for its zeros that reduces their weight on the final rendering. Since variance is often used to determine the directions in which the data will be projected for visualization, choosing a value that minimizes variance seemed to be the most relevant choice for replacing the zeros. We therefore choose to use mean imputation, where the logarithms of zeros are replaced by the average of the logarithms of the other non-zero values of the same feature.

3.3 . Visualization of aggregated data

Thanks to all the preprocessing and the logarithmic transformation, we can now apply classical methods of data visualization. This is generally done by embedding or projecting the data into a low dimensional vector space, of dimension 2 or 3 for 2D or 3D rendering of the data.

We review the different way to build such visualization and their trade-offs. We applied them to SeLfiE and HPSS and conclude on the difficulty to get a clear visualization of events data.

3.3.1 . Choice of the embedding method

It is very common that the dimension of the vector space in which numerical data are considered is too high to plot the samples directly. A visualization can only be 2 or 3 dimensional, this means that some information will be lost if we plot data that have more than 3 features, which is that case for both SeLfiE and HPSS datasets after preprocessings. The choice of a visualization method depend on what type of information we are willing to lose or preserve in the result.

We sort visualization methods between two extreme cases : linear projections and neighborhood-based embeddings. We then describe the advantages and drawbacks of methods by describing them for the two extreme cases. We choose to use linear projection methods to preserve the whole shape of the data.

Main features of an embedding method

Visualization methods can be broadly classified into two categories, each with their own advantages and disadvantages : projection methods and neighborhood-based methods.

The projection methods require the data representation to be a linear projection via a matrix decomposition of the data matrix. We find in these methods those based on the SVD decomposition such as the truncated SVD often used for latent semantic analysis, the PCA, the NMF. Additional redundant dimensions deduced from the patterns can also be added and removed to obtain more parsimonious representations as is the case in dictionary learning.

Neighborhood-based methods use instead the distances between points, and in particular the neighbors of each point, to build a non-linear latent space into which the data are projected. The most popular methods of this type are the Self-Organizing Maps, or Kohonen map, the t-SNE or more recently UMAP.

Some methods can be more or less similar to linear or neighbor-based projection methods depending on the parameters used. The kernel SVD or kernel PCA use a positive definite kernel and can be closer to a neighbor-based method if the kernel used is Gaussian or to a linear projection method if the kernel is linear or polynomial of low degree. The idea is to do a linear projection in higher dimensional vector spaces (like RKHS, Reproducing Kernel Hilbert Space, in the case of kernel based methods) where the coordinates are non-linear functions of the coordinates of the data matrix.

Description of the trade-offs in visualization

Visualization of data into lower dimensional plan necessarily involves trade-offs, information will be lost and it may be difficult to understand how the final representation can be misleading.

The main trade-off is between matching the overall shape of the data in the lower dimensional visualization with a shape that is well present in the original data and respecting the distances between the points in the visualization plane with the full data. In other words, either the shape of the visualized data is well present in the original data but the distances between the points in the visualization are not representative, or the opposite is true.

In general, linear projection methods return a visualization whose overall shape corresponds to a shape present in the original data but may artificially bring together points that are supposed to be arbitrarily distant according to the weights of the dimensions retained in the final projection. The choice of a linear projection method is therefore often dependent on well-defined

assumptions : Truncated SVD is concerned with the directions of the data with respect to the origin, PCA with the axes of the greatest variance, NMF applies to positive data and visualizes the data as a positive sum of basis vectors.

In contrast, neighbor-based methods distort the data but the distances between neighboring points are more representative. However, it is often difficult for these methods to ensure that the distance scales between widely separated points are respected. Neighbor-based methods generally have much more complex parameters to handle and interpret to manage this trade-off with distant points, they can take many forms : kernels for kernel-based methods, perplexity for t-SNE, a number of neighbors for UMAP or a metric to measure distances in general. It is not easy to know what the appropriate values of these parameters are and whether some interesting visualizations are in fact only the result of a wrong choice of them.

In our case, we do not know very well what to expect in the data due to lack of expertise. We have to choose between methods for which any observed patterns results from the original data but it is possible that the most interesting patterns could be completely erased by the projection which only focus on a shape of the data with little interest and methods that certainly return patterns that are very difficult to imagine but may be a simple artifact of the visualization method and a bad choice of parameters. For lack of guarantees, we prefer not to take the risk of over-interpreting the data and to stay with methods that facilitate the interpretation of the visualization and do not backfire by making us imagine artificial patterns. We therefore choose to focus on methods based on linear projections.

3.3.2 . Visualization of SeLfiE data

The simplest and most used linear projection method to visualize data is the Principal Component Analysis, so we train it on SeLfiE data. We use the executable filename field of SeLfiE data to approximately check if the visualization is relevant to form clusters of jobs with the same filename, this also leads us to train a Linear Discriminant Analysis mode with a regularization to avoid overfitting. Finally, we use the trained models to visualize the SeLfiE data of all the processes (without rebalancing by executable filename). The PCA components are easier to interpret than LDA components in general.

Principal Component Analysis on balanced data

After converting all extensive values into intensive values and aggregate values per processes, we assumed in the previous section that the SeLfiE data are compositional data with multiplicative noise and that the log function should therefore be applied to them before a classical statistical treatment that generally considers the noise as additive. We also felt that the calculation of a logarithm of a zero should be replaced by the average value of the logarithms of the same coordinate for the other samples when it is non-zero.

Finally, we want to use a linear projection method. Since the origin does not play a particular role after the application of the logarithm, there is no reason to use a truncated SVD method. It is therefore simplest to apply Principal Component Analysis to the results of all these preprocessings, especially since the mean imputation does not change the mean vector in case the values imputed to the logarithms of zeros are ignored.

The result can be seen in Figure 3.6. The points are colored according to the name of the file extracted from the command path field of SeLfiE data. In most cases, it can be a significant label of the class of a process. We can notice a shape in the projection over the two first components. The samples appear to be dispersed in a non-convex hull that has the shape of a cross with its

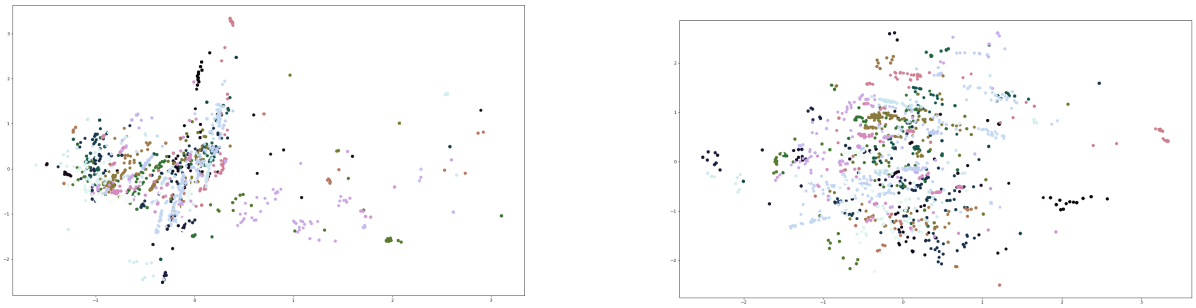


Figure 3.6 – Result of Principal Component Analysis on transformed balanced SeLfiE data
Left : Projection on the first two principal components
Right : On the second and third components

axes slightly inclined with respect to the principal components and one of its branches running in the direction of the first principal component. One possible interpretation is that the position of a sample in the SeLfiE data is mainly determined by its coordinate along the principal component when it is non-zero. To make a practical interpretation, we need to look at which combination of coordinates the principal component corresponds to, these are shown in Figure 3.7.

We show the coordinates of the first to third principal components which account for 60% of the total variance. The principal component accounts for 35% of the variance and is mainly determined by the number of function calls of the MPI library while ignoring the time spent processing them and compares it to the RAM usage (information carried by `maxmem`) and the standard I/O of the storage system (information carried by `posixio` and partly system time `stime`). The second component contrasts the time spent reading from and writing to disk storage, the number of threads and processes used and the number of instructions per cycle with memory bandwidth, RAM utilization, and the balance of system time and user time. The third component contrasts the number of reads and writes to disk storage and system time with memory bandwidth, RAM usage, parallelism (via the number of processes and threads), number of instructions per cycle and user time. It is not surprising that the MPI I/O library does not explain much of the variance in the data as few jobs use it, in most cases this coordinate contains the imputed mean value, which does not contribute to the total variance.

To simplify, a projection of the SeLfiE data that explains 60% of the variance is determined by first looking at the number of MPI calls, then comparing the time spent in reading/writing storage with the intensity and amount of RAM usage, and finally comparing the number of interactions with storage with memory usage.

So picking up on the observations about the shape of the projected data, it would seem that if a job makes frequent calls to MPI functions, then the average number of MPI calls is going to differentiate it fairly well from the others, otherwise you have to use the other data and in particular how it interacts with disk storage compared to RAM.

This last conclusion must however be moderated. The principal component carries only 35% of the variance. The first three components only carry 60% of the total variance, which is an indication that 3 dimensions are not enough to represent the distribution of the data and therefore separate the jobs according to categories if they exist. However, it is possible to make better use of the names of the commands to try to obtain a low-dimensional representation that allows the

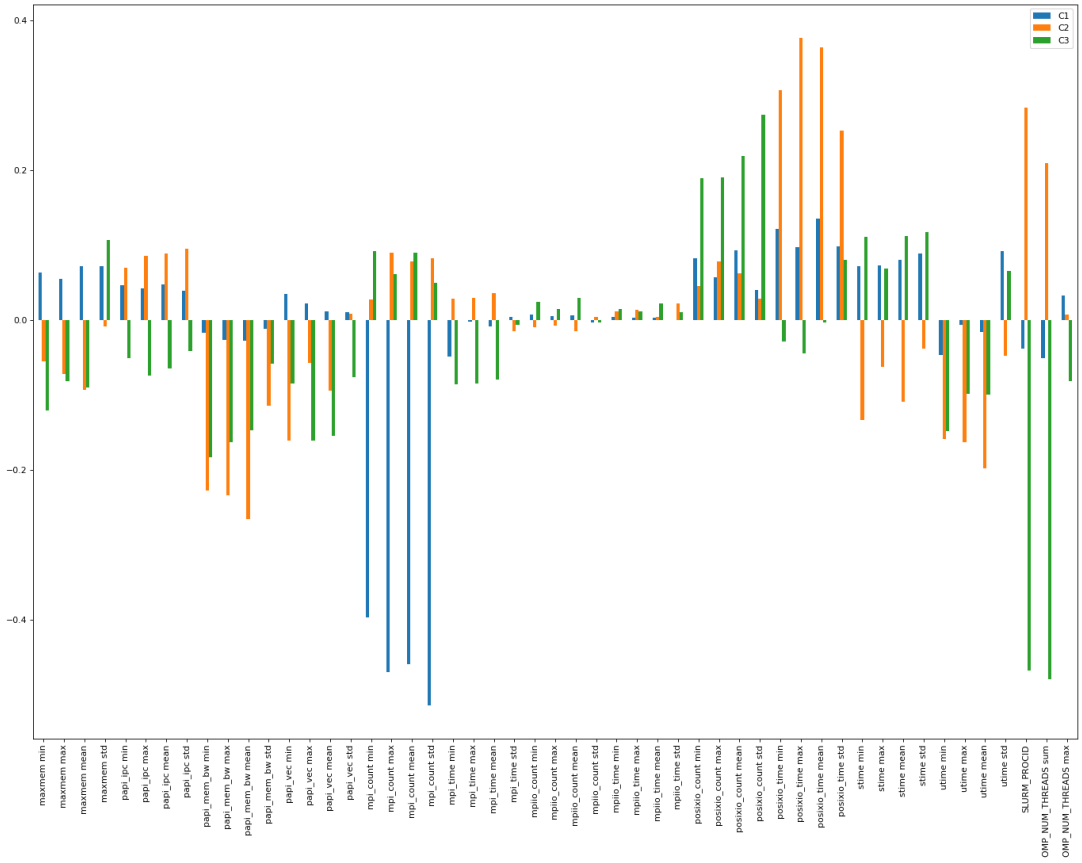


Figure 3.7 – Weight of each aggregated coordinates for the three first principal components after logarithmic transformation. The value of the projection on one of the principal components can be interpreted as a comparison of positive and negative weighted coordinates as a ratio.

jobs to be better categorized.

Linear Discriminant Analysis on balanced data

The PCA only uses the distribution of the samples and does not try to separate them in order to find classes. Thus, we were able to color the samples according to the name of the executable and observe how this variable could be distributed in the projection given by the PCA, but this data was not used at all to carry out the projection.

There are however methods that allow to find a linear projection that will rather try to separate the different classes as much as possible. The names of executables cannot constitute real job classes because they are chosen by the users at their convenience. Several executable names can be associated with the same code (LMDZ production code is often deployed in different applications which will have executable names such as `orchidee` or `opa`) and several codes can all have the same executable file name (as is the case for codes using interpreted languages such as Python, it would be necessary to know the other arguments to differentiate them). We rename the former to a common name and remove the latter from our dataset in an attempt to have a projection that is minimally distorted by this inconsistency when we are aware of it.

Linear Discriminant Analysis (LDA) is the most common method for determining a linear combination of the continuous coordinates of samples in certain classes that allows them to be separated. LDA assumes that the samples of a class are distributed according to a Gaussian distribution with the same covariance matrix (homoscedasticity assumption), so the decision boundary between two classes is linear. In practice, LDA applied to dimension reduction in the case of several classes is searching the coordinates that maximize the Mahalaobis distance between the classes instead of the principal components of the raw data as in the case of PCA. This optimization is solved by applying the eigendecomposition on the multiplication of the precision matrix (inverse of the covariance matrix) of the intra-class samples with the inter-class covariance matrix.

The direct application of the LDA method on the raw data is not a good idea when the number of features is high. This is because the number of samples per class is 50 and there may be repetition of the same sample for some classes with very few samples in the raw data, while the number of features is 47, which makes the estimated intra-class covariance matrices close to the singular. Thus, its inversion needed to calculate the intra-class accuracy matrix is not stable resulting in a lot of variance of the estimator and thus overfitting.

To reduce the over-interpretation of the LDA in the high dimensional case there are two approaches. First, one can perform a PCA with a sufficiently small number of components and apply the LDA on the projected data. To choose the number of components for the PCA according to the percentage of variance explained by these components, the most common practice is to choose a number of components that retains 90% of the variance, which is 11 components in the case of SeLfiE data. The main drawback of this method is that the key features that allow jobs to be separated may be compressed by the PCA if it has a low contribution to the variance but this is also what allows the projection on LDA components to preserve the overall shape of the data if it is useful. Another method is to regularize the estimation of the intra-class covariance matrix to reduce the variance at the cost of a bias, as done for a Ridge regression for example. The most common method implemented in SciKit-Learn library is the shrinkage between the empirical estimator of the covariance matrix and the isotropic variance, the optimal shrinkage parameter is then given by the lemma of [Ledoit and Wolf, 2004].

The result of the two methods are shown in Figure 3.8 and Figure 3.9. The classes seem less

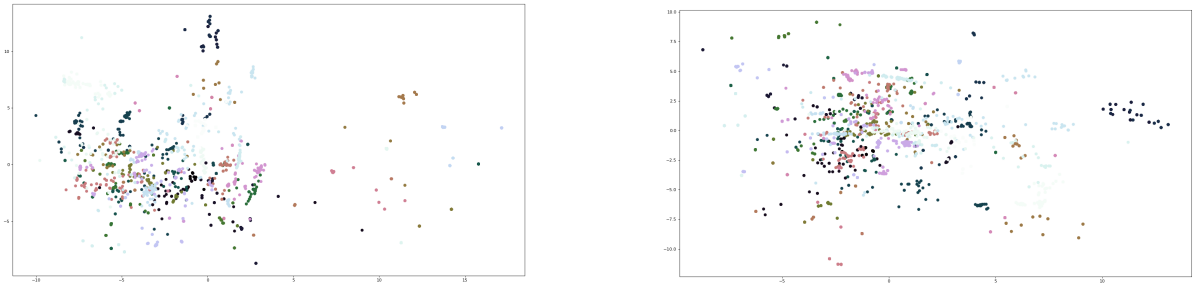


Figure 3.8 – Result of Linear Discriminant Analysis on transformed balanced SeLfiE data projected on the first 11 principal components which represent for 90% of the total variance
Left : Projection on the first two partial LDA components
Right : On the second and third components

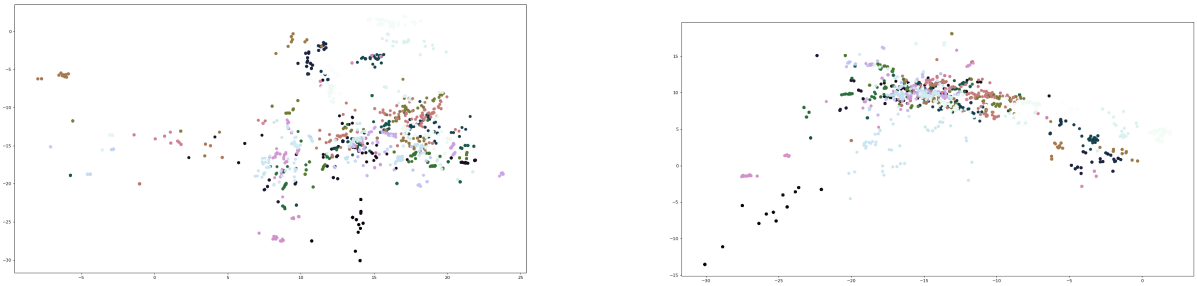


Figure 3.9 – Result of regularized Linear Discriminant Analysis on transformed balanced SeLfiE data
Left : Projection on the first two regularized LDA components
Right : On the second and third components

mixed on the projection in both cases than the PCA. Using the PCA before LDA makes the projection closer to the shape of the PCA, particularly when looking at the projection on the first two components. Shrinkage LDA returns a widely different shape which looks like a fork in 3 dimensions.

The fact shapes are so different can be explained easily by looking at the coordinates of LDA components in both cases. The coordinates of LDA components on data projected on PCA components are very similar to the PCA components themselves, particularly for the first component of LDA while the components of the regularized LDA are widely different and may use very differently the different estimator of the same data aggregated by processes as shown in Figure 3.10, this makes the result hard to interpret for the regularized LDA. We believe that the difficulty in interpreting the components of the regularized LDA suggests that it overfits the data. It is possible to obtain poorly reproducible results on some coordinates of the regularized LDA components because of the random resampling to balance the number of samples per class used in pre-processing.

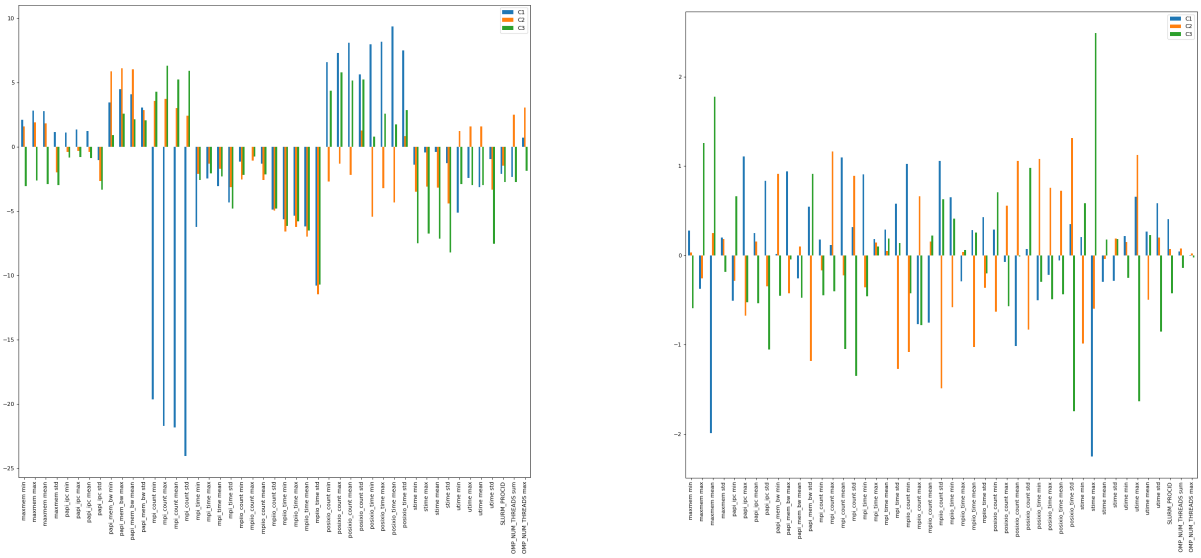


Figure 3.10 – Weight of each aggregated coordinates for the three first components given by Linear Discriminant Analysis after logarithmic transformation. The value of the projection on one of the components can also be interpreted as a comparison of positive and negative weighted coordinates as a ratio.
Left : Coordinates first three LDA components after projection on first PCA components
Right : Coordinates first three regularized LDA components

Component Analysis on all processes

Another advantage of visualization techniques based on linear projections is that they can make predictions on new data without needing to be re-trained, this is called an inductive learning. In contrast, visualizations based on neighbors are often transductive, they do not provide a way of visualizing the data but propose a visualization for the data presented to it. If one wishes to visualize new data, one must then re-evaluate the entire visualization on all the data at one's disposal and there is no guarantee that the result of previous visualizations on the data one already had will be preserved.

To ensure that each potential class of jobs is well represented we resampled the data so that each executable file name represents the same proportion of the data to create the visualization. Since we opted for an inductive modeling based on linear projections, we can therefore observe the visualization of the original data where some executables are called much more often than others as shown in the Pareto plot in figure 3.5. The results of the LDAs with PCA or regularization are shown in Figure 3.11. It seems the regularized LDA is slightly better at separating the different classes candidates of jobs but this result does not exclude a possible over-fitting since the data shown are strongly dependent to the data used to train the model. The LDA obtains after PCA projection seems to also perform well and also deserve further study and more practical testing because the components are easier to interpret than regularized LDA on aggregated data directly.

3.3.3 . Application to HPSS data

As made with SeLfiE, we can apply PCA on HPSS logs data. We first choose how we aggregate them into count data and apply the logarithm transformation. Then we looked at projection on

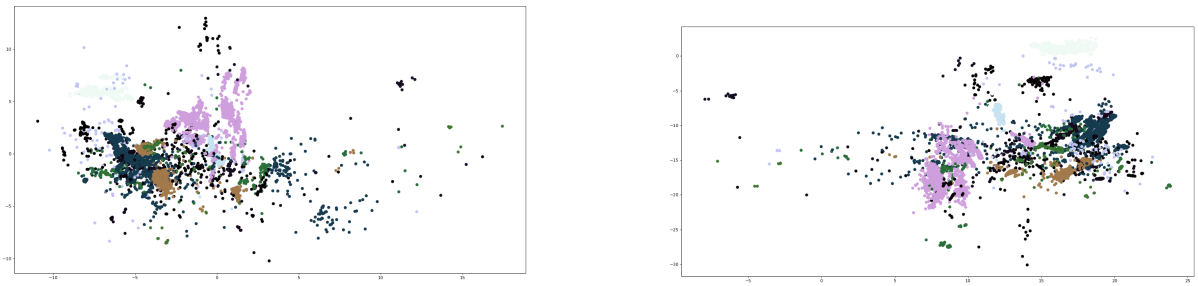


Figure 3.11 – Projection of log-transformed SeLfiE data without resampling to have the same number of samples with the same executable filename on previous Linear Discriminant Analysis components.
Left : Projection of original data on the first two LDA components on data projected on the first principal components
Right : on the first two regularized LDA components

the principal components while keeping track of the order of counts samples to judge if we are able to discover relevant time periods. Finally, it seems that our work does not apply well on HPSS logs when compared with SeLfiE results.

Choosing the aggregating criteria

Before proposing a visualization similar to the one we did for SeLfiE, we first need to choose the aggregation criterion to transform the raw HPSS data in the form of an event list into a count time series. The aggregation criterion is a pair of time period and field value. We choose to aggregate the events over 30-minute periods in order to have few periods where no events are emitted (95 periods out of 2724) while remaining quite short of incident durations that are difficult to detect in practice.

The severity and type of events are data that have few distinct unique values and inform more of the interest in reading the content of the message than in differentiating the failures. Message identifiers have many unique values and it is expected that the number of unique values will increase with the number of samples. This would quickly become a problem when generalizing the visualization obtained to larger datasets. This is why we make the compromise of using the names of the routines issuing each event as field values, these allow us to know what HPSS was doing in a time period and which tasks generated the event record without knowing what the event is. The number of routines called in our dataset is 121, which is comparable to the number of features used to visualize the SeLfiE data.

The resulting count data is 121 time series, one for each routine, of 2724 time periods.

Visualization with PCA of HPSS data

The time series of HPSS event counts can also be considered as compositional data since each component is the number of events emitted by a routine and is therefore proportional to the proportion of events emitted by that routine during the period. Therefore, we also have to take the logarithm to transform the multiplicative noise into additive noise in order to use a PCA as for the SeLfiE data. However, we do not have to convert count values which are extensive into intensive values because the time aggregation criterion is the same size for all the points.

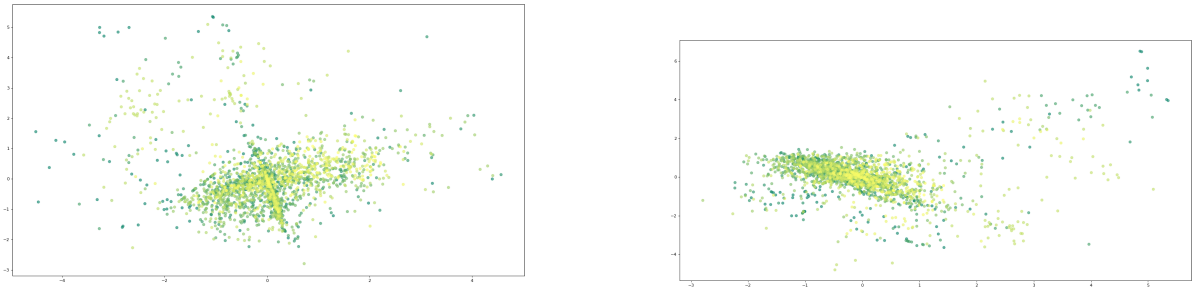


Figure 3.12 – Result of Principal Component Analysis on log-transformed HPSS data
Left : Projection on the first two PCA components of HPSS data
Right : On the second and third components

Since we want to use the PCA, it again seems more reasonable to use an average imputation to give a value to the logarithm of the zeros so that they do not arbitrarily distort the final representation. The result of the PCA is shown in Figure 3.12. The points are colored according to the date of the time period they represent, the oldest are green and the most recent are yellow. Contrary to the SeLfiE data, no particular grouping can be distinguished, almost all the points are in a group which does not present any particular structure, apart from a greater density of points on one of its axes which can certainly be explained by the presence of zeros on a set of coordinates for these even if they are not out of the ordinary. In some periods the points deviate from this central group and return to it without forming a particular structure.

The coordinates of the first three principal components are shown in Figure 3.13. We can see that the routine `ns_Find` defines most of the principal component and `tp_GenericRead` weights a lot in the definition of the second and first principal components. The coordinates of the first principal components are mostly the routines emitting most of the events.

Open problems with HPSS data

Once in the form of a time series of counts, HPSS data remains quite different from SeLfiE data. There is no class that our visualization should try to preserve as we did with SeLfiE data with LDA. This also implies that we cannot resample the data to make it representative of all the events that may happen as we did for SeLfiE before visualization. A good visualization of HPSS data could also preserve a proximity between two periods close in time but there is no way to our knowledge to enforce this constraint on the result of visualization. This could be done with convex optimization by adding a penalty to the objective function associated with PCA that favors visualization that keep close point in time close in the visualization space but it goes beyond the focus of this work.

Another big difference is that a feature in HPSS does not have the same role as in SeLfiE. In SeLfiE, more data will not result in more features. In HPSS, collecting more logs implies that there are higher chances that we observe rare logs from rarely emitting routines and therefore increases the number of features to consider. This means a visualization of HPSS data with the proposed method is not guarantying to be relevant after some time and may have to be trained again.

While SeLfiE data and HPSS count data can both be considered compositional and require applying the logarithm before any processing, HPSS data are much more challenging because of

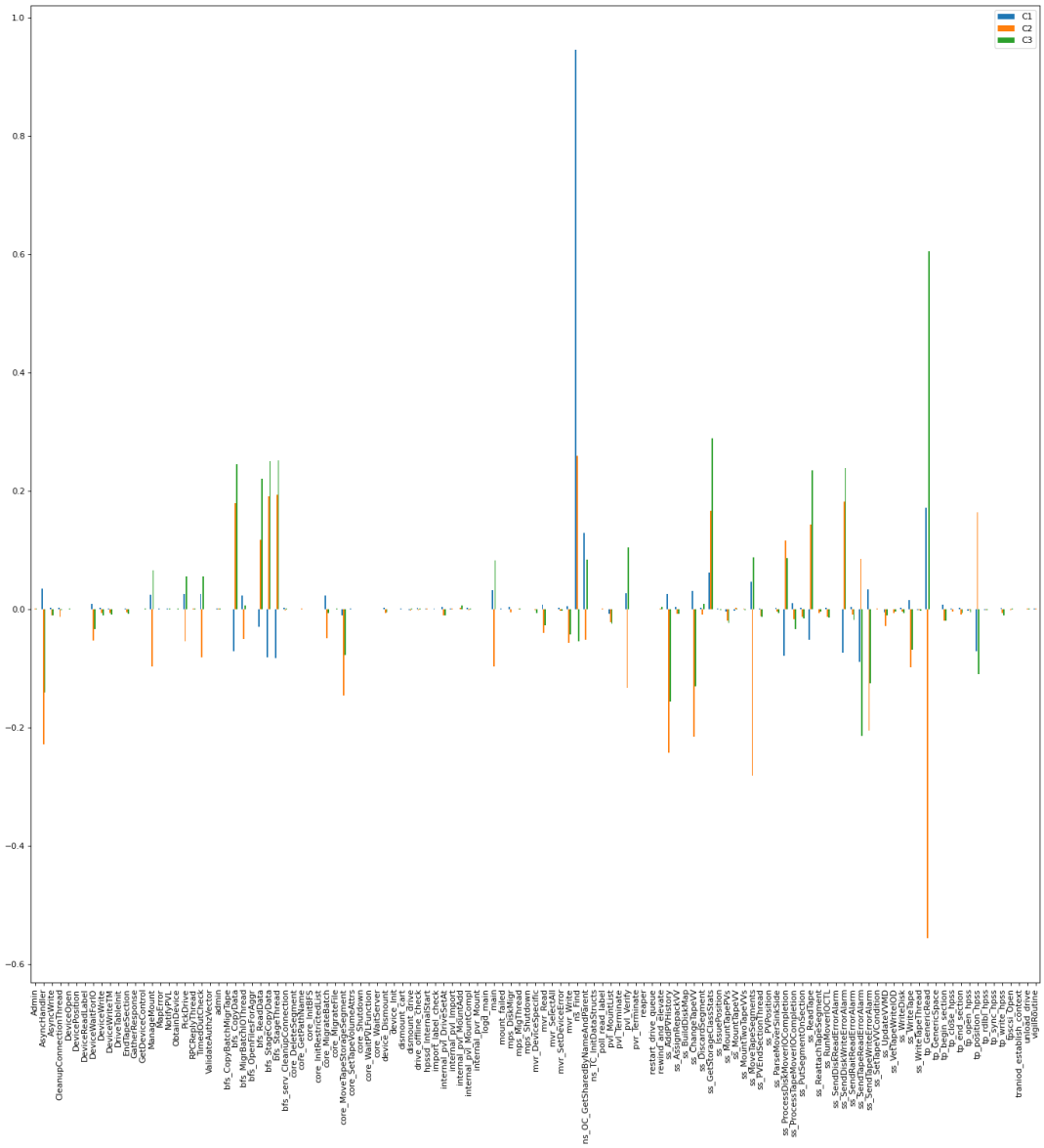


Figure 3.13 – Weight of each type of count for the three first principal components after logarithmic transformation of HPSS count data. The value of the projection on one of the principal components can be interpreted as a ratio of occurrences between positive and negative weighted coordinates powered by the absolute value of the weight.

the lack of labeled data and unlimited number of events type.

3.4 . Conclusion on treatment of events data

The logs are the data that contain the most valuable information on the operation of the HPC center because they are the only data that gives direct information on the tasks performed by the different software by default and without any particular intervention.

However, we have seen that the lack of structure in this data format to identify a type of event makes it very difficult to exploit. This problem could only be solved with an expertise or even a change of the data format to include such an identifier.

In cases where such an identifier is available (SeLfiE and HPSS), we have presented what is the data and proposed to consider the aggregation of events by a simple count according to criteria imposed by the data extraction or not. The result is however quite different from the counting data in the literature on this subject as the values are either very large or zero. We used the compositional data framework to model these data, thus leading us to consider the logarithm and to look at an imputation value for the case of structural zeros.

Finally, we show the result of the PCA on the result of all these preprocessings. In the case where labels are available, we succeed in identifying groups of events that seem to be coherent and we can refine the visualization to better highlight them while keeping the ability to generalize. The result is however difficult to evaluate in the case where labels are not available. All the visualizations proposed here are only a first step and it would be risky to deduce from them methods that can be used in production at the moment. More expertise is needed to interpret the results and to collect more appropriate datasets for the calibration of such models. With well-made dataset, more complicated models could be used to add semantic in it, like GPT3 or an attention layer to isolate which events are weak signal of failure in the future.

4 - Unsupervised modelling of interconnect network nodes' communication time series by clustering and segmentation

The communication between nodes is a critical data to understand how the computation are managed on the nodes for a given job. However, the volume of such data is so big that it is impossible to collect it without modifying the code running on nodes or saturating the available bandwidth. The compromise is to aggregate the data at the node level in time and return it as a multivariate time series of bytes count for every node each 5 seconds. This raises the question : are those aggregated data still meaningful ? Our aim is to provide a way to find features that sum up the communication data of the nodes running a given job at best so that they can be used later to improve the understanding of the computation management of production sized jobs.

We first explore the data from benchmark jobs and compare them with production jobs. The exploration reveals that data must be normalized to be able to extract interesting features, we decide to work on the log-returns series. We propose a conditional modelling of log-returns.

We find that inferring the features previously observed is actually a subspaces clustering. We describe a family of models for time series subspaces clustering and provide efficient solvers based on convex optimization to fit them. A numerical experiment of the two simplest models of this family on a small dataset shows which model we expect to perform better on our data.

However, we found that this model adds too many constraints on the resulting segmentation, so we propose another model with an efficient method to fit it. We check that fitting the model gives the expected results and propose heuristics to set hyperparameters. We also notice that there are cases where the heuristic to find the value of model's hyperparameters cannot be applied, which suggests again changes in the model for future work.

4.1 . Modeling raw communication data

We first quickly recall what are raw bytes' communication data and start exploring its main features on known and unknown examples. This leads us to search a method to normalize the data so that we can extract features we observe during exploration and propose a model.

4.1.1 . Exploration of the data available

Raw bytes' communication data has not been well documented before. Therefore, we have to carry an exploration, first on benchmarks which are known jobs, then on unknown production jobs. This allows us to identify interesting features to extract.

Exploration of data from known benchmarks

The network card of each computing node counts the number of bytes send and received every 5 seconds. The data is then collected and aggregated into multivariate time series, one for each job, through the pipeline described Figure 1.4 of Chapter 1. We decide to focus only on the

incoming bytes on each node and not the outgoing to simplify. The incoming bytes also carry information that will be used by workers while outgoing bytes can be directed to other auxiliary hardware like the storage for record. For a given job, the communication data is the evolution of the number of bytes received by each of the node allocated for this job from the time the nodes are allocated to the end of the whole job script execution. The communication data is then a set of integer-valued and regular time series, one for each node. All the time-series also have the same size as the measurement are aggregated on the same period.

We have data from known benchmarks that are regularly run to check that the HPC center is still performing well after maintenance. These benchmark jobs are well known. We also have SLURM scheduler log information about them, so we know how many steps the job contains, when it starts and when it stops and the name of the executable file launched. We also know how the computation is structured in terms of both parallelism and the nature of the data processed. It is thus possible to link certain events to observations on the raw communication data. All benchmark jobs have only one step.

The data from our benchmarks are shown Figure 4.1. The benchmarks are the following :

xHPCG : It is a benchmark used to rank computing centers. The complete description of the toy problem solved is described by [Heroux et al., 2013] as "a single degree of freedom heat diffusion model with zero Dirichlet boundary conditions" on a regular mesh using a conjugate gradient solver which assign a cubic region to each MPI process. The resulting time series are shown at the top left of Figure 4.1. It seems the amount of bytes received by a node in average during the computation depend on the number of region neighbors of its assigned cubic region (it can be any integer between 3 and 6 the amount of bytes received during 5 seconds is obtained by multiplying the number of neighbors by 5MB)

ABINIT : Abinit is a production code which initially solves density functional theory equations [Gonze et al., 2020] but also support many other quantum physics related computations. It requires solving linear algebra problems, often a generalized eigenvalue problem, or performing Fast Fourier Transform. No mesh is required. The resulting time series of a benchmark example are shown at the top right of Figure 4.1. The logs from SLURM shows that the Abinit executable is launched lately and this can be noticed by the long period of low raw bytes communications. We also see that all nodes have the same communication load and we can observe a repeated pattern which may correspond to some iterations.

AVBP : AVBP is a production code to solve Navier-Stokes equations mainly used to simulate fluid dynamics and combustion in engines. It is often used on non-regular meshes. The resulting time series of a benchmark example are shown at the bottom left of Figure 4.1. We can observe big spikes of incoming bytes before and just after the executable file AVBP is launched respectively on one unique node and then all of them. It may correspond to the loading of the mesh into the RAM. Once loaded, some nodes seem to always receive more bytes than others in the same proportion but all the nodes seems to follow a common pattern except for the scale.

YALES2 : YALES2 is a production code similar to AVBP but is more focused on two-phases combustion problem. The resulting time series of a benchmark example are shown at the bottom right of Figure 4.1. The behavior of the time series seems to be very similar to the AVBP case except that the spike of incoming bytes to load some data before actually running the core computation are lower but longer. It seems that it took a more significant amount of time to load data into the RAM of every node before the computation. We can

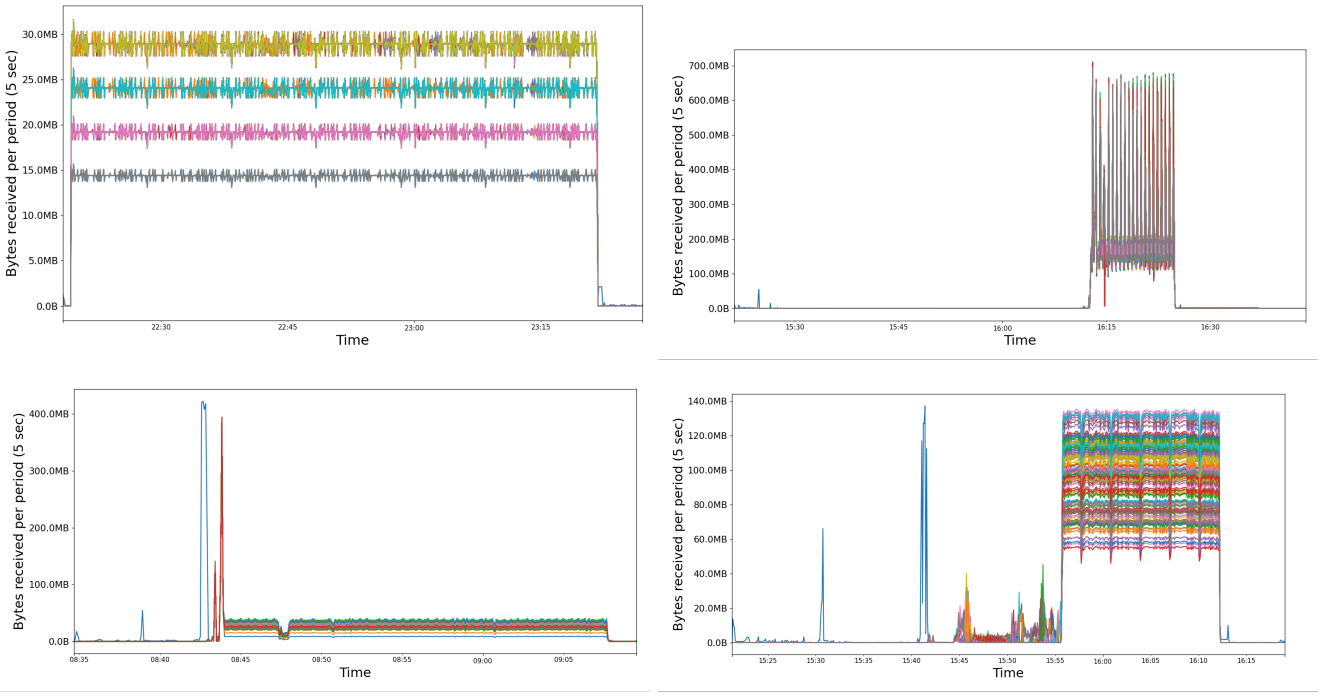


Figure 4.1 – The raw incoming bytes communication data measured for 4 jobs running different benchmarks.
Top row : xHPCG (left) and ABINIT (right)
Bottom row : AVBP and YALES2 (two industrial codes for fluid and combustion simulation)

not determine if it is related to the meshes or YALES2 code. After loading, there are more communications between nodes and as in the case of AVBP, the incoming bytes load is not balanced between every nodes.

It seems raw bytes communication data behaves like continuous random positive real data despite being integer-valued for at least two reasons. First, the order of the integers is high enough to mask discreet behavior. Second, the time period of 5 seconds to aggregate of the incoming byte events is so high that we cannot consider the underlying event process is stationary during it. This may explain why the variance is much higher than the average number of counts contrary to what we expect from a Poisson law.

Nodes tend to all follow the same pattern except for the scale when the core computation is running, this is however not necessary the case before and after such period. We suspect it may be associated to load of data on which the computation will be run (and record of the result at the end). The scale of the amount of received bytes may provide information on the meshes used for the computation, it seems we can clearly differentiate regular from non-regular meshes. We should also be able to determine the load imbalance between the nodes with this data alone in case of non-regular meshes. This may also explain that a code solving problems where there is no mesh has an almost perfect load balance between nodes.

So it might be possible to get more information on the code running on nodes from raw bytes' communication time series if such features also appear in real production job and if we can extract them consistently.

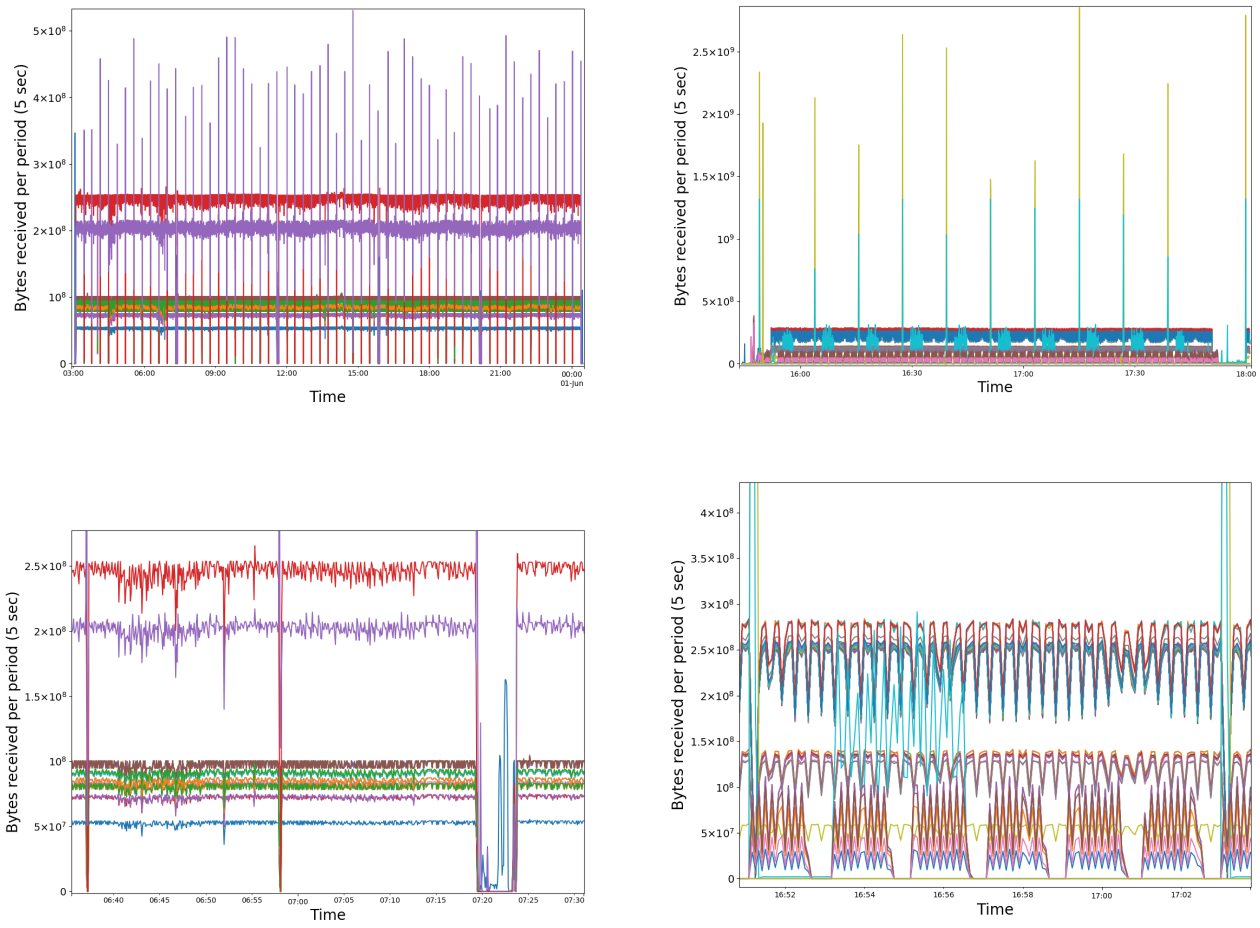


Figure 4.2 – Measurement from unknown production jobs used as examples. The plots at the top show the whole series, the bottom ones display a zoom on a repeating pattern.
Left : A job using 32 nodes over one day.
Right : A job using 189 nodes over 2 hour and a half

Exploration of data from production

The previous benchmarks are used to check that the HPC center is performing normally. We do not know if the data is representative of the running jobs of the computing center in production context. When users run their jobs, it is possible that several jobs run on the same nodes. We have no way to detect it reliably without more data from SLURM logs. The jobs in production also have several steps : several executables can be run several times. The users' jobs are also not perfect, the load unbalancing can be bigger and we have no reason to think that the imbalance will be preserved during the whole job duration. To get an idea of what pattern can be present in production jobs data, we need to take a look at unknown jobs.

We find that the raw communication time series of many jobs are very similar to the ones from benchmarks. But we focus on two examples that retain our attention shown in Figure 4.2. Each column shows the evolution over time of raw communication received by each node allocated during the same job, the complete evolution is shown at the top row with a zoom on interesting time periods for better observation at the bottom.

We can make some observations from the data shown in Figure 4.2.

- The features observed in benchmark jobs can be found in the production jobs.
- There is a periodic pattern that does not look like a simple computational iteration. There are regular spikes of extreme values. One or two nodes received suddenly a lot of bytes while all the others a lot less. This happens for 1 to 10 seconds in general. It suggests a check-point during the job, intermediaries results are written on disks and it can be long enough to be noticed. However in the first presented case, there are also much longer period of several minutes every 4 hours when only one or two nodes received bytes while all the others received a lot less as shown in the zoomed period Figure 4.2 at the bottom-left.
- It seems that there is a very strong correlation between groups of time series during what seems to be the core computation periods. We can visually gather nodes which follow the same pattern up to scale into groups. Contrary to the benckmarks, we must consider several groups of nodes and several periods of intensive correlated communications. These periods can be common to all groups as in the first case or not as in the second case where a group of nodes seems to stop communicating with each other six times more frequently.
- There are regularly periods of low communications that may look like check-pointing moment but are much longer. This may be associated with transition to a new job steps.

The two examples we showed were the most different from the data available. We can find the same features we observed in the benchmarks' data with the difference that we may have to consider groups of nodes instead of all of them. We also found out that we may have to consider that the core computation period is split into several regular periods separated by check-point or transitions to a new step.

4.1.2 . Data normalization with log-returns

We chose to work on the log-returns of the series to remove the scale locally. Then we look at how the log-returns behave during the different phases we identified previously.

Local normalization with log-returns

[Etienne and Latifa, 2014] proposed a model for count time series obtained from arrivals counting that follow a Poisson process of variable intensity. They derived that an appropriate normalization is to divide each time series by their mean. However, the regular spikes of extreme value of some nodes when others are close to 0 suggest that the mean will be biased higher for the former. An alternative method could be to use a robust estimator of central tendency like the median but such estimators work by ignoring part of the data. It is possible that most of the time points are not relevant, for example in the bottom-right plot of Figure 4.1, which makes such estimator discarding the period where we observe the scaling between series. There is no "global scaling factor" by which we can divide the series to scale them correctly.

We propose to work on the log-returns of our time series instead. The log-returns are the time series obtained by taking the successive difference of their logarithm. They are used in financial data analysis to remove any influence of the price scale of different assets.

We denote the whole communication data as $X = \{X_1, X_2, \dots X_N\}$ where N is the number of nodes used by the job and X_i is the time series of bytes counts received by node i every 5 seconds.

Given the time series $X_i \in \mathbb{R}^{T+1}$ of length $T+1$ the series of log-returns $S_i \in \mathbb{R}^T$ are computed as :

$$\forall t \in \{1, \dots, T\}, s_{it} = \log(x_{i(t+1)}) - \log(x_{it}) = \log(\frac{x_{i(t+1)}}{x_{it}})$$

We can immediately notice that the series X_i and X_j have the same log-returns series if and only if there is a positive real α such that $X_j = \alpha X_i$. This means log-returns are indeed a form of data normalization that remove the effect of scale.

Behavior of the log-returns

The log-returns of our previous examples are shown in Figure 4.3. We observe that log-returns are matching almost exactly for nodes with the same proportional communication main pattern during what seems to be the core computation period. However during low communication phases, the log-returns have a very high variance and the series don't match anymore. Nodes still received bytes during low communication phase with several order of magnitude less. However, the communication are much more random at this lower scale as the variance of the logarithm is a lot higher during those phases.

This confirms that the log-returns are good candidates for local normalization of our communication series. Because they can be computed as the logarithm of the series of successive ratios, it agglomerates series together if they are proportional at a given period. However, the low communication phases are still a problem because the volatility (variance of the log-returns) is high since nodes seems to randomly communicate. This suggests that we could use log-returns to cluster nodes if we are able to exclude high volatility period before or during clustering.

4.1.3 . Modelisation of communication log-returns

From our exploration of log-returns, it seems they have well-defined phases according to their variance and they match exactly when time-series are proportional. We propose a simple gaussian conditional model to generate log-returns for which we know the phases and clusters. This allows us to simulate samples that look like log-returns for which we know the ground truth for clusters and phases.

Gaussian conditional model of log return

To specify a conditional model of log-return, we suppose that we are given latent time series W_k valued in $\{0, 1\}$ for each cluster k corresponding to the assignment of a point in time to what we believe is a core computation period for a group of nodes or not. We also assume that we are given means series Z_k for each group k.

Our conditional model of log-return is following : if the log-returns series Y_i is in cluster k and W_k is equal to 1 at time t then its value at time t is the value of Z_k at time t plus a centered white noise of standard deviation σ. If W_k is equal to 0 at time t instead then the log-returns series Y_i value is draw from a centered white noise of standard deviation σ' where $\sigma \leq \sigma'$.

This model corresponds to a simple mixture of 2 gaussians weighted by the value w_{kt} :

$$p(y_{it}|Z_k, W_k, i \in Cluster_k) = w_{kt}\mathcal{N}(z_{kt}, \sigma) + (1 - w_{kt})\mathcal{N}(0, \sigma' I) \tag{4.1}$$

Because it is only a conditional model, it doesn't make any assumption on the prior distribution of W_k and Z_k. To generate data, we need to specify how they are chosen.

Simulation of log-return

To simulate a dataset of log-returns with the previous conditional model (4.1), we can set the series of weights W_k so that each component is regularly in the phase where $w_{kt} = 1$ for a fixed

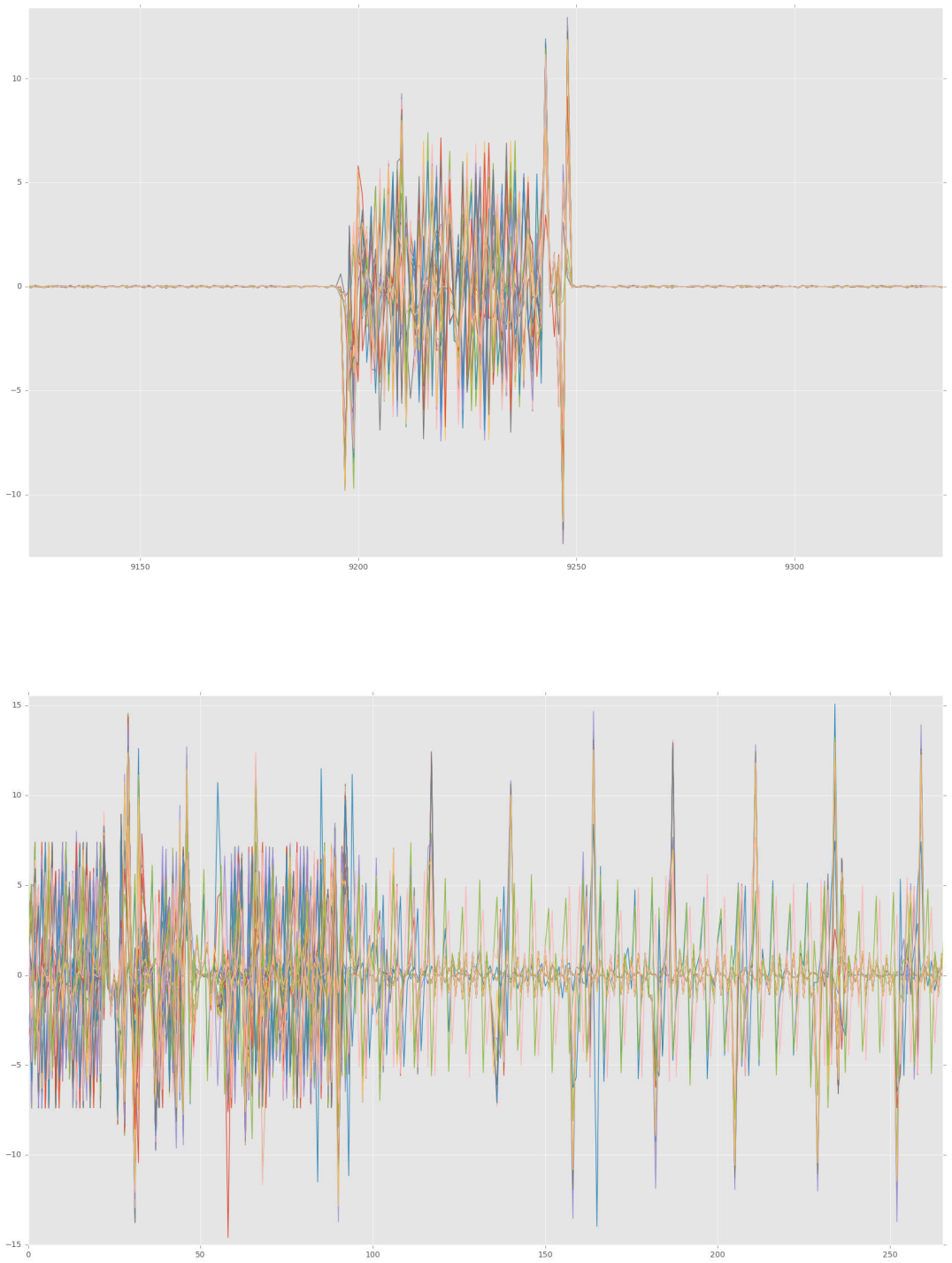


Figure 4.3 – Zoom on parts of the log-returns series of Figure 4.2. The scale invariance of logarithm aggregates the value of proportional series at the same time. They reveal that the communications are much more random when they are low, this is highlighted by the significant increasing of variance of log-returns also called volatility in the financial industry.

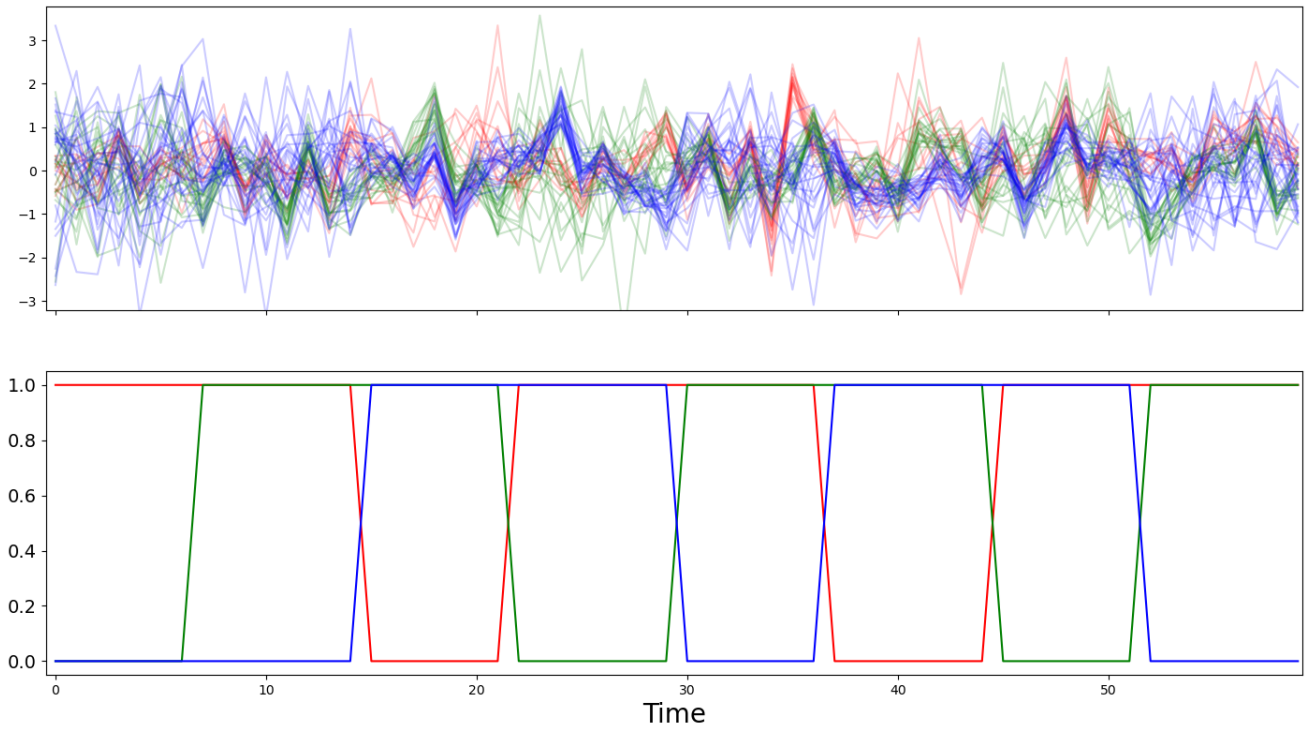


Figure 4.4 – 60 Simulated series data of length 60 with 3 clusters. The weights' series of each cluster are shown below. Synchronized phases of each cluster overlap over 1 period.

duration. The samples can be assigned to any cluster uniformly. We choose $\sigma' = 1$ and $\sigma = 0.1$.

Once the above parameters with correct size are given, a data simulator generates N sample series of length T in K clusters which by sampling conditionally the distribution (4.1) using the normal distribution. An example is given in Figure 4.4.

The result is fairly representative of the communication log-returns of a true job as plotted in Figure 4.3. So our goal is now to derive inference methods of the parameter of this conditional model. This corresponds to perform a subspace clustering.

4.2 . Models for Time series Subspace Clustering

For the rest of this chapter, the time-series considered are only the log-returns of raw communication data received by nodes.

We propose a generic cost function that can be used to perform a subspace clustering of time series with a penalization and describe the fitting algorithm. We focus on a particular type of penalization and propose a solver for this type that is faster in cases already seen in the literature thanks to convex optimization. Finally, we compare how the two main penalization functions of this type perform.

4.2.1 . Time series Subspace clustering algorithms

We present what is subspace clustering. We propose a model for subspace clustering of time series. Then we show how it can be fitted.

Subspace Clustering

Subspace clustering is a class of clustering algorithm where points in the same cluster are not necessarily close for a given metric but close to a common subspace for this metric. This class of model is very large depending on assumptions we make on the subspaces. Subspaces can have any dimension depending on the clusters, they can be affine space like in generalized PCA [Vidal et al., 2005] or not.

Some subspaces clustering method only consider axis-parallel subspaces. Clustering with feature weighting [Huang et al., 2005] is a subset of this class. The aim of such method is to assign weight to features of each cluster to lower or even remove the contribution of some of them that do not matter for clustering. In our case, the features are time points. We would like to ignore time points only where log-returns are not agglomerated.

The main feature of time series is that their value at different time points are not independent. We expect that the closer the time points to each other, the more dependent the value of the time series at these points are.

Time smoothing of subspaces

When a clustering with features selection is applied on a time series dataset, the weights of clusters are also time series. [Huang et al., 2016] propose to enforce the time series that describe the selection of features of each cluster to be smooth. We formalized in a more general cost function minimization the subspace clustering of time series :

$$P(U, Z, W) = \sum_{k=1}^{K} \left(\sum_{i=1}^{N} \sum_{t=1}^{T} w_{kt} u_{ik} d(x_{it}, z_{kt}) + \mathrm{pen}(W_k) \right) \tag{4.2}$$

under the constraints : $\begin{cases} \forall i \in \{1, 2, \ldots, N\}, \sum_k u_{ik} = 1, & u_{ik} \in \{0, 1\} \\ \forall k \in \{1, 2, \ldots, K\}, \sum_t w_{kt} = 1, & w_{kt} \in [0, 1] \end{cases}$

d is a distance or dissimilarity measure in the value space. K is the number of clusters we want to have, N is the number time series in the dataset and their length is T.

The results are returned in U and W. U is the assignment matrix of each series with a cluster k ($u_{ik} = 1$ is equivalent to the series i being in cluster k). The $(W_k)_k$ are the positive weights time series summing to one of each cluster k. Time series $(Z_k)_k$ are centroids of groups k depending on the dissimilarity measure chosen.

pen is a penalization function to enforce the $(W_k)_k$ series to have a desired regularity when minimizing $P(U, Z, W)$.

Model fitting

The model (4.2) is a particular case of K-means if d is the square of the euclidean distance and pen a characteristic function of constant series (meaning $\mathrm{pen}(W_k)$ is $+\infty$ if W_k is not constant). This implies that (4.2) is at least as hard to optimize as K-means, for which the exact resolution is infeasible [Lloyd, 1982] because it is a hard combinatorial problem.

We can approximate a solution with a classical alternate optimization of each variable with others fixed to attain at least a local minimum of the objective function. The assignment matrix U is initialized randomly and the weights W are set as constant series equal to 1. We fix sequentially every variable other than $(Z_k)_k$, then U and finally (W_k) and iterate the three steps until convergence is achieved. The convergence is reached when the assignment matrix U stay still after an iteration. One iteration is decomposed in three smaller optimization problems :

Z Step The membership U and weights W are fixed and we optimize the centroids Z. The terms in the sum over k are independents so that we can compute the series $(Z_k)_k$ separately. In the most common case d is the square of euclidean distance and the centroid Z_k is just the series of averages of the time series in group k. A particularity of subspace clustering is that z_{kt} can be any value if $w_{kt} = 0$.

E step The means series $(Z_k)_k$ and weights $(W_k)_k$ are fixed and we optimize the membership express in the binary matrix U. Because the terms in the sum over k are independent, each of this term is minimal at optimum. So we assign each series X_i to the cluster k for which $\sum_{t=1}^{T} w_{kt} d(x_{it}, z_{kt})$ is minimal. Notice also that the choice of the value of z_{kt} when $w_{kt} = 0$ may alter the new membership. If this step did not change the value of U, convergence is reached and we stop the optimization.

W Step The assignment matrix U and the centroid series are not fixed and the weights' series $(W_k)_k$ are updated.

The Z and W steps require their own optimization procedures. The Z step is often simple to solve because the solution is known to be the average for the most common choice of d which is the square euclidean distance. However, the W step is harder to solve because there are generally no way to express solution obtained from a regularization other than the square of euclidean norm that is not useful in this case.

4.2.2 . Resolution for convex gradient-based cases

We present penalization on the weights time series on the successive differences to make them regular. Then we propose a very efficient solver for the W step for such penalization choices.

gradient based penalization

The model 4.2 require specifying two main parameters : the dissimilarity d and the penalization function pen. The cases where d is not the squared distance are much harder to solve so we will fix d to be the squared euclidean distance. This means the Z step is just computing the average series of each time series in the current cluster.

We want to choose a penalization that translate the fact that close time points are more dependent to each other. We want to introduce an explicit chronological dependency in the optimization model by choosing a penalization pen in model 4.2 that reduce the variation of the weights when they are close. A simple way to enforce this is to penalize the high values of the series of successive differences of weights' series (that we will call the "gradient of the weight series").

We can use the squared euclidean norm of the gradient as a penalization. This corresponds precisely to what [Huang et al., 2016] proposed when they used the Sobolev energy of the weights' series $(W_k)_k$ as penalization :

$$\mathrm{pen}(W_k) = \frac{\alpha}{2} \sum_{t=1}^{T-1} (w_{k(t+1)} - w_{kt})^2 = \frac{\alpha}{2} ||BW_k||^2 \tag{4.3}$$

with $B \in \mathbb{R}^{T-1 \times T}$ the matrix associated with the computation of successive differences :

$$B = \begin{bmatrix} -1 & 1 & 0 & & & \\ 0 & -1 & 1 & 0 & & (0) \\ & \ddots & \ddots & \ddots & \ddots & \\ (0) & & 0 & -1 & 1 & 0 \\ & & & 0 & -1 & 1 \end{bmatrix}$$

so that

$$BW_k = \begin{bmatrix} w_{k1} - w_{k0} \\ w_{k2} - w_{k1} \\ \vdots \\ w_{kT} - w_{k(T-1)} \end{bmatrix}$$

It is well known that Sobolev energy minimization enforce smooth solution that have the regularity properties of the result of a low-pass filter or heat diffusion [Calder et al., 2010]. It is also well-known [Evans and Gariepy, 1991] that the squared euclidean norm can be replaced by the L1 norm of the gradient to obtain piecewise-constant solutions if desired. α is a hyperparameter that handle the trade-off between the regularity of the weights' series and the precision of the selection of features.

This lead us to define a gradient-based penalization as a penalization function pen that can be written in the following form :

$$\mathrm{pen}(W_k) = f(BW_k) \tag{4.4}$$

where f is any proper convex function on $\mathbb{R}^{T-1}$.

We need to solve the W step efficiently at each iteration of the three steps of the main alternate optimization.

Solver for gradient based penalization

From (4.2) and (4.4), we deduce that gradient-based penalization require to solve a W step of the following form for each cluster k :

$$\begin{cases} \min_X & Q_k^\top X + f(BX) \\ \text{s.t.} & X \in \Delta_T \end{cases} \tag{4.5}$$

where Δ_T is the probability simplex $\Delta_T = \{X \in \mathbb{R}^T \text{s.t.} X \geq 0, \mathbb{1}^\top X = 1\}$ and the $(Q_k)_k$ are defined as the series local variances of the time series in each cluster k since d is the square euclidean distance :

$$\forall t \in 1, \ldots, T, q_{kt} = \sum_{i=1}^{N} u_{ik} d(x_{it}, z_{kt}) = \sum_{i=1}^{N} u_{ik} (x_{it} - z_{kt})^2 \tag{4.6}$$

[Huang et al., 2016] proposed to use a quadratic solver like `cvxopt.solvers.qp` of CVXOPT library [Vandenberghe, 2010] in the case where the gradient-based penalization is the Sobolev energy (4.3). Indeed, calling `cvxopt.solvers.qp`$(\alpha B^\top B, Q_k, \mathbb{1}^\top, 1, -Id, 0)$ exactly solves the W step for Sobolev energy. However, the solver is slow and cannot be used for other gradient-based penalizations. The parameters (except Q_k) of the quadratic problem have particular structures which are not fully used to optimizer the solver.

Instead, we propose to use convex optimization to solve (4.5) much faster. The constraints of the problem (4.5) can be removed by adding the convex characteristic function δ_{Δ_T} as penalty to the objective function. For any convex set A, its convex characteristic function δ_A is defined as follows :

$$\delta_A : x \mapsto \begin{cases} 0 & \text{if } x \in A \\ +\infty & \text{else} \end{cases}$$

The updated value of each W_k is then the solution of the following convex problem :

$$\min_{X \in \mathbb{R}^T} f(KX) + g(X) \tag{4.7}$$

with g is a convex function (recall that f is also convex by definition) and K the linear operator defined as :

$$\begin{cases} K & : X \mapsto BX \\ g & : X \mapsto Q_k^\top X + \delta_{\Delta_T}(X) \end{cases}$$

Problems of the form (4.7) are particular cases of convex duality theory known as Fenchel-Rockafellar duality. Convex duality provides very fast algorithms to solve convex optimization problems. Details on the convex duality and optimization procedures for Fenchel-Rockafellar duality can be found in Annex A. There are several requirements before being able to use optimization procedures based on convex duality.

We need to compute the convex conjugate (defined in the Annex A) of all the functions involved and the dual of the linear operator K. It is easy to derive that $g^* : X \mapsto \frac{1}{\alpha} Q_k + \max_{t \in \{1,\dots,T\}} X_t$. The image of a series $Y \in \mathbb{R}^{T-1}$ in the dual space by $K^* = Y \mapsto B^\top Y$ is the vector of successive difference with a first and last supplementary coordinate respectively equal to the opposite of the first coordinate of Y and the last.

We need that a saddle point solution of the problem (A.2) exist, meaning we want to have primal and dual feasible points. $(1/T, 1/T, \dots, 1/T)$ is in the relative interior of $\mathrm{dom}(g) = \Delta_T$ and its image by K is the null series in $\mathbb{R}^{T-1}$. The relative interior of $\mathrm{dom}(f^*)$ contains also the null series as soon as f has a minimum, which is generally the case for a penalization function, so $(1/T, 1/T, \dots, 1/T)$ is primal feasible. The domain of g^* is the whole space $\mathbb{R}^{T-1}$, we can also expect f^* to be at least proper, so any point in its domain is strictly dual feasible. The two conditions of existence of a saddle point of the form (A.2) are verified (details on sufficient conditions for the strong duality can be found in Annex A).

Finally, we need to be able to compute the proximal operators of f^* and g. The proximal operator of g can be obtained directly by including the scalar product with Q_k into the squared norm.

$$\begin{cases} \mathrm{prox}_{\tau g}(X) & = \mathrm{argmin}_{Y \in \Delta_T} Q_k^\top Y + \frac{1}{2}||X - Y||_2^2 \\ & = \mathrm{argmin}_{Y \in \Delta_T} \frac{1}{2}||(X - Q_k) - Y||_2^2 \\ & = \mathrm{Proj}_{\Delta_T}(X - Q_k) \end{cases}$$

The projection on Δ_T is a threshold of excess-mass above 1. Fast computation of this projection have been already proposed by [Wang and Carreira-Perpinán, 2013].

Thanks to all these analytics expressions, we can use the Chamboles-Pock algorithm from [Chambolle and Pock, 2011] detailed in Annex A to solve any W steps for a gradient-based penalization where f is convex proper function with a minimum. The solver's parameters (σ, τ) can be

chosen so that $\sigma\tau < \frac{1}{4}$ because $||K||^2 < 4$. The parameters σ and τ are often set equal and the largest possible by default, this means we could choose them close to $\frac{1}{2}$ but slightly lower.

However, we repeat this optimization several times with a vector Q_k that will be very different at the beginning of the clustering algorithm and at the end. The number of iterations n_{iter} needed to have a sufficient convergence can be very different and if the optimization algorithm did not converge enough at each update of the weights the whole clustering may not converge. To avoid having to handle parameters while keeping the number of iteration low enough for every time W step, we use the linesearch method of [Malitsky and Pock, 2016]. Details on the optimization procedure with linesearch can be found in Annex A.

4.2.3 . Two gradient-based penalization comparison

We check that we reproduce the result of [Huang et al., 2016] when we use the Sobolev energy as penalization. Then we compare with the total variation.

Results with Sobolev penalization

Our solver can be used for the W step with $f(X) = \frac{\alpha}{2}||X||^2$ in (4.5) to produce the same subspace clustering as [Huang et al., 2016] called TSkmeans. Such f has a minimum, a known proper convex conjugate ($f^*(X) = \frac{1}{2\alpha}||X||^2$) and proximal operator ($\text{prox}_{\sigma f^*}(X) = \frac{\alpha}{1+\alpha}||X||^2$).

We simulated the data used by [Huang et al., 2016] to illustrate the smoothness of weights in the case of a Sobolev penalization. The number of series of length $T = 15$ is $N = 300$, there are $K = 3$ clusters with non-overlapping phases. When the time series of group are not following the same pattern, their values are draw from a uniform noise as well as the pattern they would follow. When they follow the same pattern, a gaussian noise of low enough variance is added. We reproduced these simulated data in Figure 4.5. Our data simulator allows us to evaluate the clustering method result on much bigger data and the scalability.

In the case where the penalization is the Sobolev energy, the resulting solutions are exactly the same as the ones obtained with the quadratic solver used by [Huang et al., 2016]. 100 iterations of the Chambolles-Pock algorithm with linesearch are enough for all iteration on all the data on which we run it, each iteration is very fast as the complexity of one iteration is dominated by the projection on Δ_T which is $O(T \log T)$ where T is the length of the time series.

To compare the smoothness of weights, the weights found by W-k-means [Huang et al., 2005] and our implementation with Chambolles-Pock solver of TSKmeans [Huang et al., 2016] subspace clustering after convergence are shown in figure 4.6. We set $\beta = 2$ for W-k-means and $\alpha = 2$ for TSKmeans. We can easily notice that each cluster has a weight series which is high only where the series of a true cluster are close of the series of average. Weights are positive for W-k-means while they can be 0 with TSKmeans on time points that are totally irrelevant for clustering, which is not the case of W-k-means weights. This gives a clear segmentation of the series in a cluster.

However, the smoothness of weights' series in TSKmeans case implies they are smaller at the edge of relevant time for a cluster. The smoothness of TSKmeans weight introduce a bias at the edges, weights are small when close to irrelevant time point. It is possible to show that the shape of the weight vector is a reversed parabola when the variance around the mean series is constant on relevant periods using the optimality conditions of our problem.

This lead us to minimize the total variation of the weight instead of Sobolev, thus enforcing weights' series to be piecewise-constant.

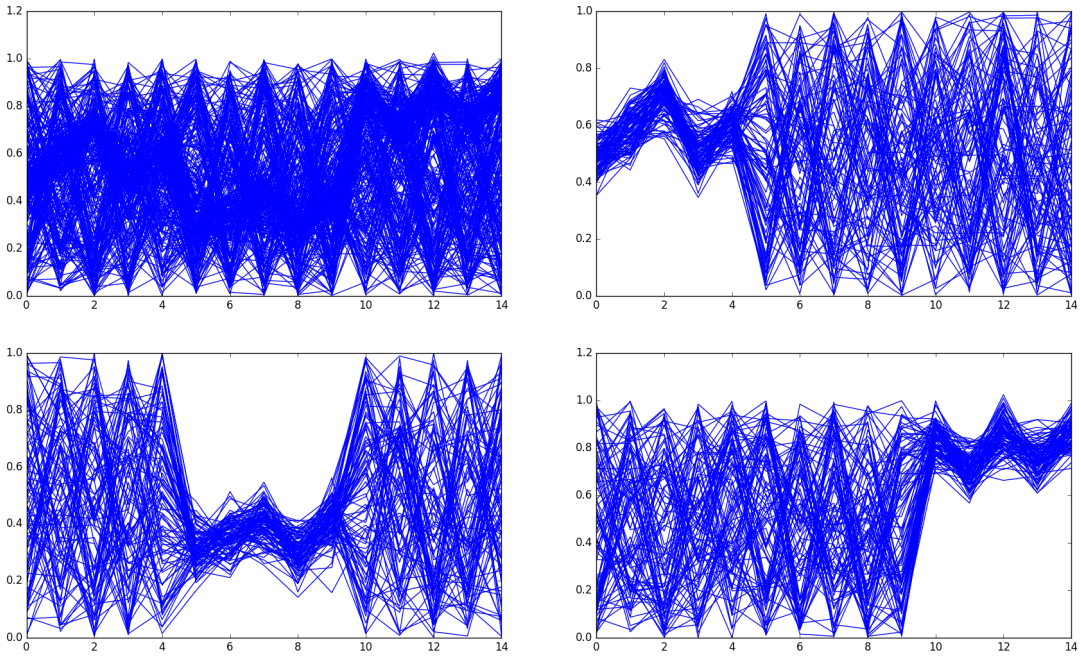


Figure 4.5 – Whole simulated dataset by [Huang et al., 2016] (top-left) and the 3 clusters

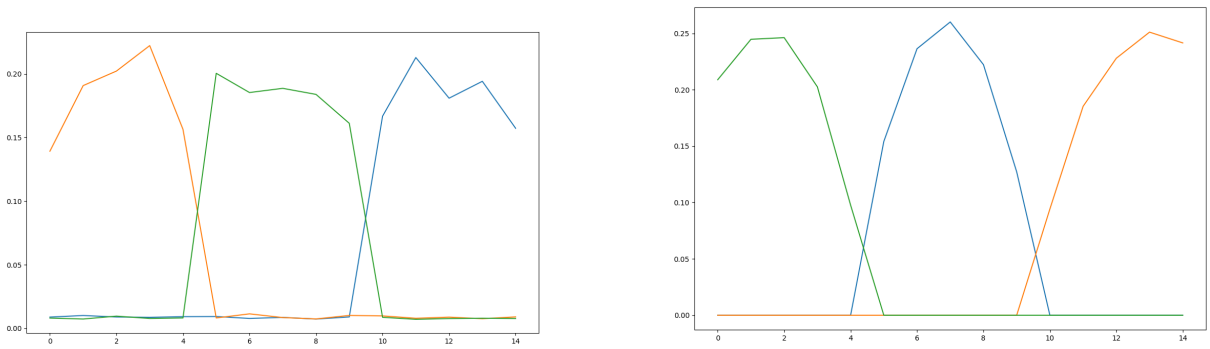


Figure 4.6 – The weights found by W-k-means and TSKmeans for [Huang et al., 2016] simulated data.

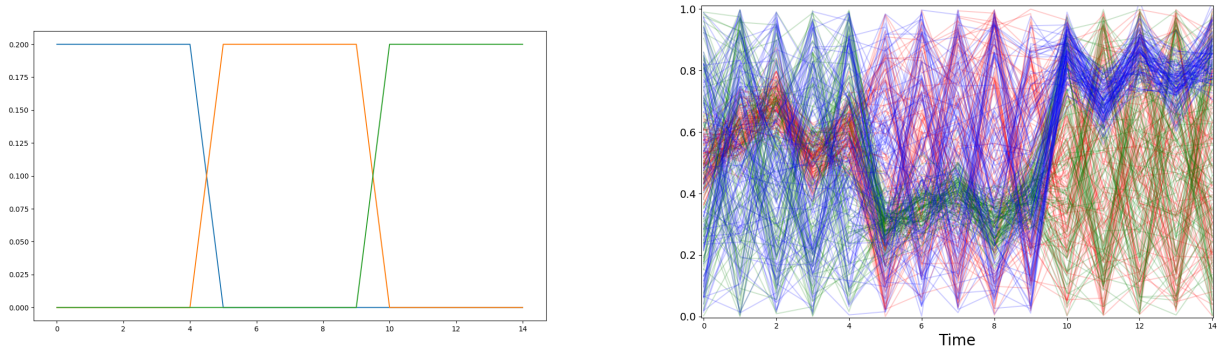


Figure 4.7 – The weights' series (left plot) found by sharp TSKmeans (total variation penalization) on [Huang et al., 2016] simulated data (cluster shown in the right plot) with $\alpha = 2$. Weights can still be exactly 0 and the weights' series are sharp, they are not smaller at the edge of relevant period for a cluster as it is the case of original TSKmeans.

Sharp penalization

Our solver can natively handle the use of total-variation as a gradient-based penalization by setting $f(x) = \alpha||X||_1$. It is a proper convex function with a minimum. Its convex conjugate is known to be $\delta_{||.||_\infty \leq \alpha}$ which is proper too and its proximal operator is the soft-threshold at level α.

To judge the effect of changing the penalization on weights' series on clustering result, we propose to compare ourselves to K-means as made by [Huang et al., 2016] with different similarity measures such as the Euclidean norm, the Pearson correlation [Liao, 2005] and the "Short Time series distance" introduced by [Möller-Levet et al., 2003] as well as the TSKmeans proposed by [Huang et al., 2016]. The quality of a cluster is expressed by the F-measure, the RandIndex and the NMI as introduced and detailed respectively by [Manning et al., 2008], [Rand, 1971] and [Strehl and Ghosh, 2002]. On 100 simulated data sets, we observe these scores and retain the average. The results are given in the table 4.1. We use the global scaling defined by [Huang et al., 2016] as value for α in TSkmeans.

Algorithm	Fscore	RandIndex	NMI
Euclid	0.5949	0.7226	0.4078
Pearson	0.5003	0.6642	0.2719
STS	0.3818	0.5870	0.0951
TSkmeans	0.9736	0.9654	0.8985
TV penalization	1.0	1.0	1.0

Table 4.1 – Comparison between clustering algorithms on 4.5 data

Our algorithm perfectly clusters the series. The shape of the weights obtained by this new model on the same dataset as Figure 4.5 are shown in Figure 4.7. As expected from the total variation minimization, the weights are piecewise-constant. The better clustering results may be explained by the fact that the series of [Huang et al., 2016] dataset do not smoothly agglomerate but very suddenly. The Chambolles-Pock solver always find the optimal solution in less than 100 iterations.

The weights also take a zero-value when the series of a group are not following a common pattern as we saw with previous model. This makes them straightforward to convert into the discrete valued W_k vectors of each mixture component in (4.1) by associating a value of 1 for any positive value and 0 for the already zero values. It is straightforward to decompose a job into small "benchmark" jobs with the result of this model.

4.3 . Non-uniqueness of the selection with total variation

Changing the penalization has another undesired property on the weights' series. To remove it, we add a new term in our penalization, for which we can use an even more efficient solver and interpret hyperparameters. We finally look at the results and find out that there are still open questions to solve in order to build a model fully matching our expectations.

4.3.1 . Identification of the uniqueness issue

Several issues can be noticed when we use the Time series Subspace Clustering model on longer series.

First, the Chambolles-Pock algorithm takes more iterations to converge. The alternate optimization is also longer and it may also end up in an infinite loop if the Chambolles-Pock required more than 100 iterations to converge.

The resulting weights of time series that have several time intervals where they are close to their common pattern also look incorrect as shown with simulated data from our conditional model in Figure 4.8. The total-variation penalization seems to allow solutions where only one segment of uniform weights per cluster is selected to be optimal, but it does not allow to have several segments as we would expect.

This can be explained by the fact that the total-variation is constant on some subsets of the probability simplex. An example of two weights' series (of length 3) with the exact same total variation is given by the two example on the right of Figure 4.9. The left panel is a potential vector Q_k obtained from other step of the whole optimization from the update (4.6) on time series of length 3. A high value of Q_k is associated with a high deviation of the series of the group considered around their mean, the weight update will return weights series that minimizes the penalization function and its scalar product with Q_k. Let suppose that the value α is set so that the middle weight is zero. Then the series of weights returned by the optimization is given at the right panel. This is because in this particular case of a zero weight in the middle, the total-variation of the weights' series is just the sum of the other two, which is necessary 1 because of the constraint that they sum to 1 to be in Δ_T (with T=3). So the total variation is constant and the scalar product is maximized by setting the weight maximal where the series Q_k is minimal. This implies that only one segment of time points (the one with the lower local variance given a fixed total variation) can be selected.

This is quite similar to the degenerate case of the W-k-means subspace clustering method of [Huang et al., 2005] but with a single segment selection instead of single feature. This can also be related to a common issue with lasso regression on strongly correlated predictors described in details by [Zou and Hastie, 2005]. The common solution is to add again an L2 penalization with its own hyperparameter giving a family of penalization called elastic-net. This leads us to once again propose another penalization.

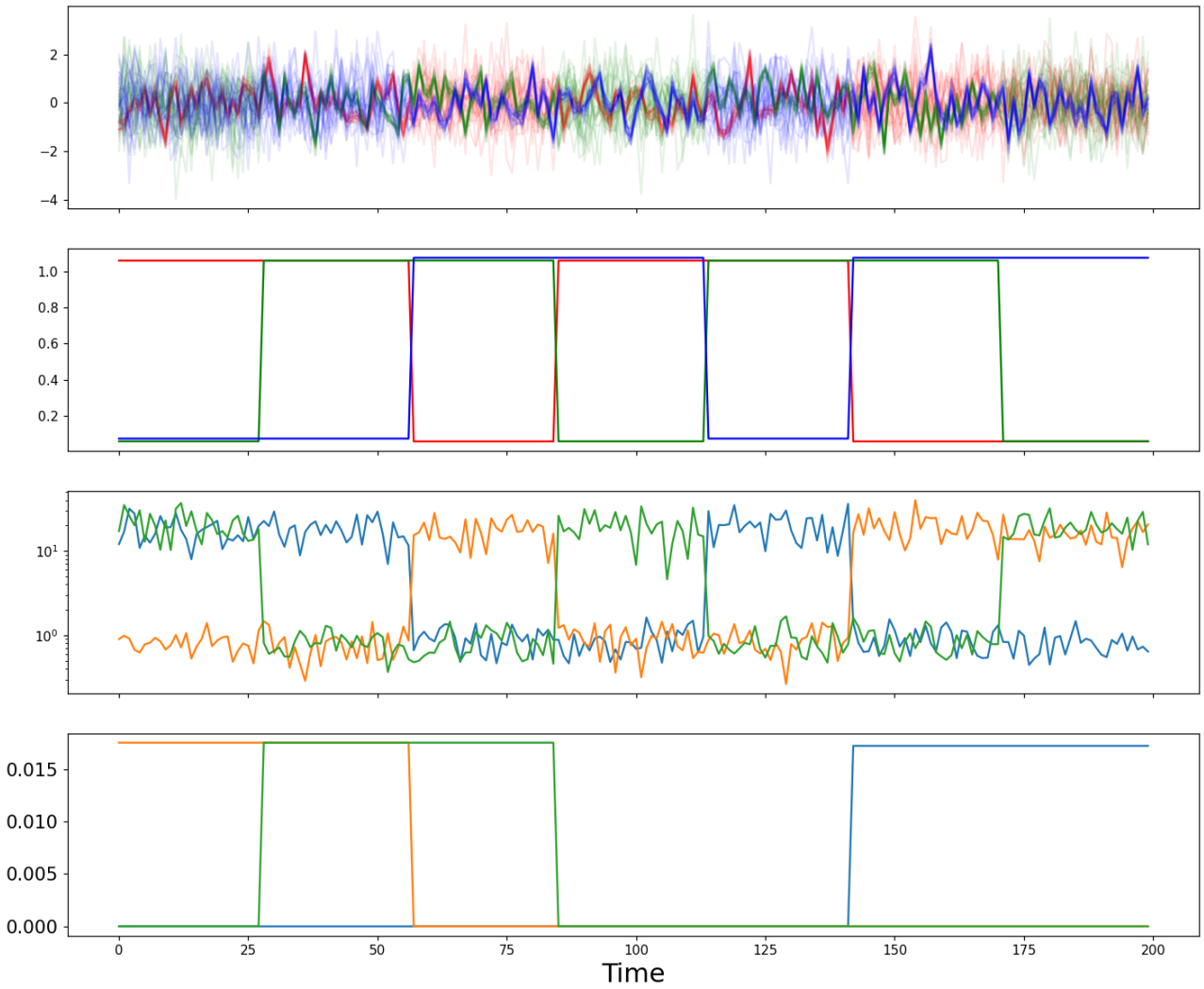


Figure 4.8 – Segment selection with sharp TSKmeans in practice.
First row : The data.
Second row : real segmentation (a small random constant is added to better read segmentation per cluster).
Third row : sum of square errors to the mean series (used to compute weights' series).
Fourth row : Weight series found, they do not cover all the periods where series in the same groups are close.

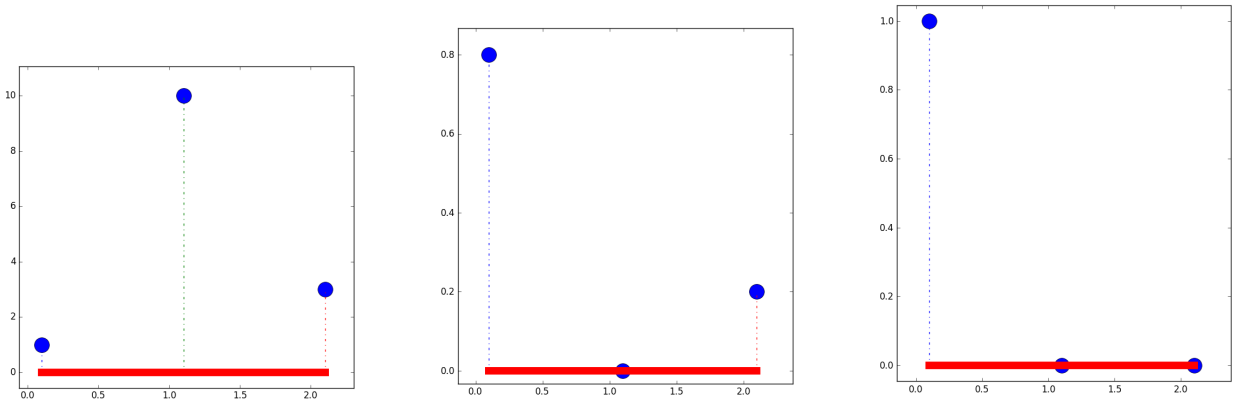


Figure 4.9 – An example of local sum of squared series and two weights' series with the same total variation. Given a Q_k vector shown at the left, the solution of the weights update is at the right while we would prefer to return the ones at the center in subspace clustering, both have the same total-variation

4.3.2 . Elastic-Net addition to total-variation

We define a new penalization which allow several time periods to be considered as relevant for the clustering. Then we show how to solve the W step since this penalization is not a gradient-based. We explain how to set the hyperparameters of the new penalization.

Penalization definition

The common approach to solve the previous problem is to introduce an L2 norm in the penalization to produce a so-called grouping effect, several variables are selected as relevant features at the same time when the penalization factor decreases. The problem of finding the form of several relevant areas for regression is also handled by [Dubois et al., 2014] by adding an L2 norm on the features vector to an already present L1 and total variation penalization. The obtained penalization is called TV-Elastic Net. Our approach is the same, we add an L2 norm on the weights vector to force the selection of several intervals in the original Sharp TSKmeans model. Contrary to [Dubois et al., 2014], we don't need to add an L1 norm penalization on the weights, this penalization is already included in our model through the simplex constraint. This means that our new Elastic penalization is defined by :

$$\text{pen}(W_k) = \alpha||BW_k||_1 + \beta||W_k||_2^2 \tag{4.8}$$

This new penalization has one more hyperparameter and is not gradient-based. However, it is still convex, so we can still find way to solve the optimization associated with the W step quickly.

Resolution with proximal operators

The W step that must be solved is the following :

$$\min_{W_k \in \Delta_T} Q_k^\top W_k + \beta||W_k||_2^2 + \alpha||BW_k||_1 \tag{4.9}$$

with

$$q_{kt} = \sum_{i=1}^{N} u_{ik}(x_{it} - z_{kt})^2$$

This problem (4.9) is much simpler to solve and does not require a primal-dual optimization as gradient-based penalizations. Indeed, we can also write it :

$$\min_{W_k \in \Delta_T} ||\frac{1}{2\beta}Q_k + W_k||_2^2 + \frac{\alpha}{\beta}||W_k||_{TV}$$

We notice that the solution can simply be express as the image of a proximal operator :

$$W_k = \text{prox}_{\delta_{\Delta_T} + \frac{\alpha}{\beta}||.||_{TV}}(-\frac{1}{2\beta}Q_k)$$

This proximal operator can be decomposed as a composition of two proximal operators.

For any X one can show that there exists λ such that $\text{prox}_{\delta_{\Delta_T}}(X) = \text{proj}_{\delta_{\Delta_T}}(X) = (X - \lambda \mathbb{1})^+$. This shows that the order of the coordinates is preserved by this proximal operator which is the sufficient condition given by [Shi et al., 2016] to decompose the proximal operator of the sum as a composition of proximal operators as follows :

$$W_k = \text{prox}_{\delta_{\Delta_T}}(\text{prox}_{\frac{\alpha}{\beta}||.||_{TV}}(-\frac{1}{2\beta}Q_k)) = \text{proj}_{\Delta_T}(\text{prox}_{\frac{\alpha}{\beta}||.||_{TV}}(-\frac{1}{2\beta}Q_k)) \tag{4.10}$$

This means that the solutions we are searching for are simply the image of a TV denoiser of a vector proportional to Q_k projected on Δ_T. TV denoising is a very well-studied problem, there is no analytical solution but direct methods to solve it. In particular in our one dimensional case, the most common method is the taut string algorithm introduced by [Davies and Kovac, 2001] which is almost linear complexity. [Condat, 2013] also proposed a direct algorithm to compute it which is linear with respect to the size of the series in practice even if it can be quadratic in some irrelevant cases. We already saw the projection on Δ_T is a threshold of excess mass and can be computed using the algorithm proposed by [Wang and Carreira-Perpinán, 2013] and its complexity is almost linear. This means each of the two operators can be directly computed in 1D in quasilinear complexity at most. The main advantage is that these methods are direct and not iterative like primal-dual methods we had to use for other problems.

The drawback of this new model is that there are two hyperparameters to tune, but they are hopefully much easier to interpret than the hyperparameters of gradient-based penalizations.

Hyperparameter settings

From the composition of the two proximal operators at the weights update step, the parameter β can be interpreted as a scaling constant between the weights and the sum of squared errors with the series of means of a group when they are close. Because the weights' series sum to 1, they are close to $1/T$ in average. So β must be chosen close to the variance of time series when they are following the same pattern times T times the number of series in the group considered. If the clusters are roughly the same size, this heuristic recommends choosing $\beta = NT\sigma^2/K$ where σ is the local standard deviation of time series when they are close to their centroids as in (4.1).

If we assume β fixed by the previous heuristic, the ratio α/β balances how smooth the weights will be according to total variation compared to the square of L2 norm of the weight. So we must set this ratio so that it is proportional to the frequency of time periods we expect to have times the

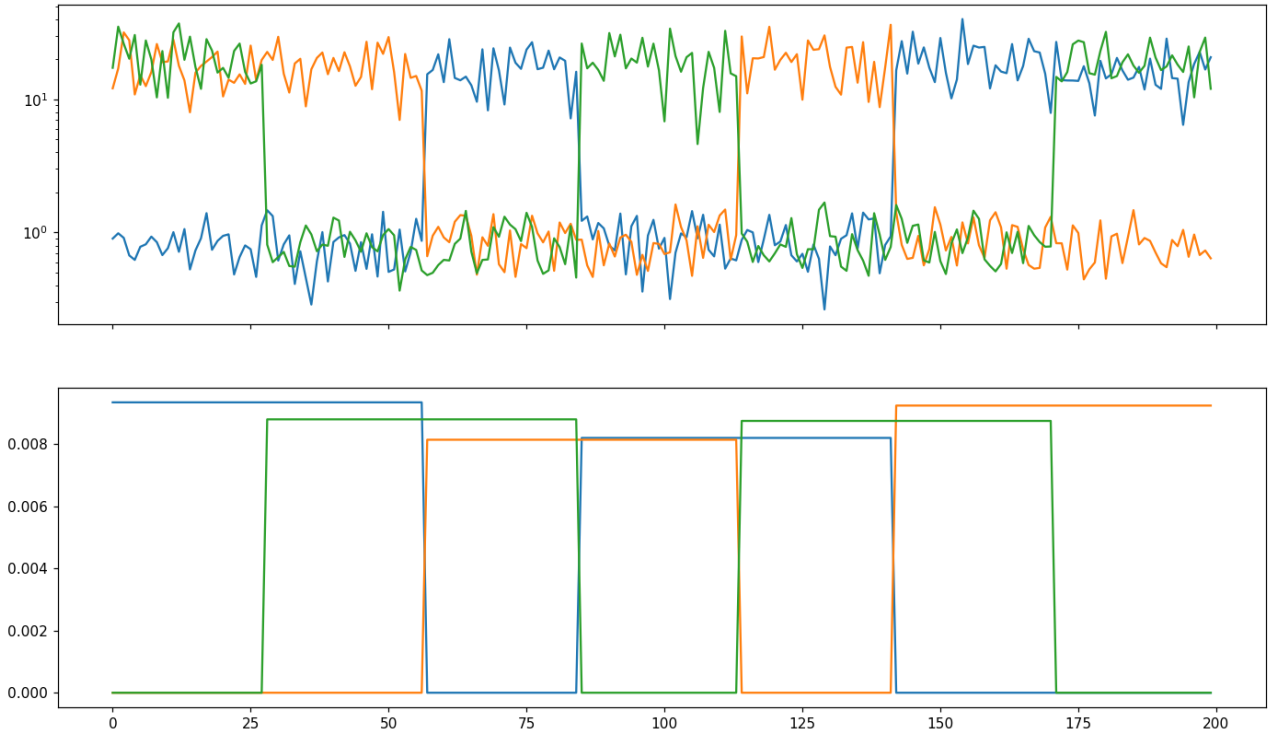


Figure 4.10 – Sum of squared errors with means series of groups and weights found with grouping effect for the same data as Figure 4.8. Notice that weights are positive on several segments, they are also higher when the segment is at one of the ends of the series

average value of the weights which is at least $1/T$ (or simply non-zero). If we expect L time periods where series are close to their centroids, then this heuristic recommends using $\alpha = 2L\beta/T$. In practice, we noticed that if β is set correctly, the range of values of α that does not change the result is large.

These heuristics assume that the weights' series are close to $1/T$ when their values are not 0 but this is a clear underestimation since they must sum to 1. So we recommend using small higher multiples of the values given by the heuristics in the hyperparameters of the model.

4.3.3 . Results

Finally, we present the results on simulated data and the real job data that motivated this work and propose futures directions to make the model more reliable in practice.

Simulated data

We show the application of this new model on data simulated for Figure 4.8 with the parameters given by the previously described heuristics. The clustering is once again correct in the showed example, but we notice that there a lot less failure to cluster simulated series and the main algorithm is much faster. The weight series looks like a piecewise-constant approximation of the opposite of the sums of squared error to the means plus a constant, which is a direct consequence of (4.10). It is easy to read which segment are used to cluster the series by looking at positive values of the weights. The segments at the ends of the series used for clustering tend to have higher weights because they contribute only once to the total variation of weights' series

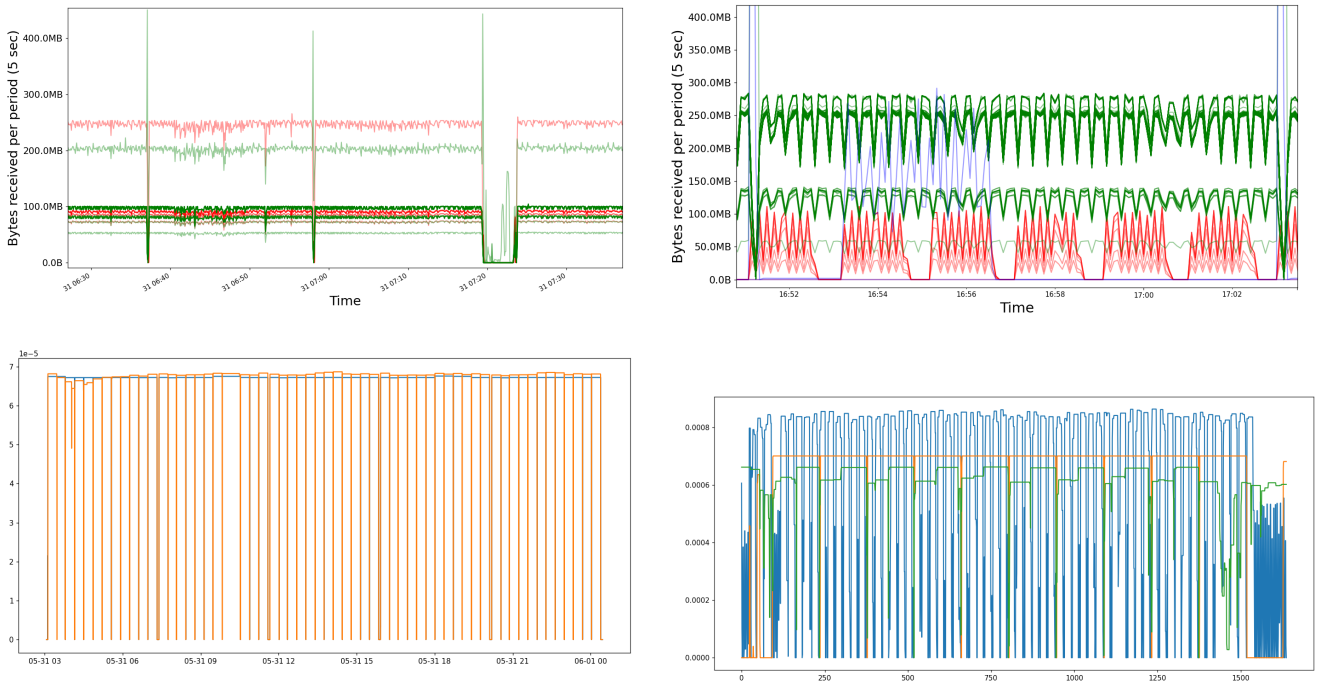


Figure 4.11 – Results of the sharp TSKmeans clustering with grouping effect on the log-returns of examples and weights' series. The top row shows plots of series on a repeating pattern according to cluster label and the bottom row shows the weights' series
Left : Job using 32 nodes over one day with $K = 2$, $\alpha = 0.1$ and $\beta = 22500$.
Right : Job using 189 nodes over 2 hour and a half with $K = 3$, $\alpha = 3$ and $\beta = 22000$

while segments inside implies two changes of values.

Real job data

The results of our model on the production jobs plotted in Figure 4.2 are shown in Figure 4.11. We use $\alpha = 0.1$ and $\beta = 22500$ on the job running on 32 nodes and cluster nodes in two groups, and we use $\alpha = 3$ and $\beta = 22000$ on the job running on 189 nodes and cluster nodes in three groups. The hyperparameters correspond to higher values than our heuristic to better handle the largest group of series we expect to find. In both shown cases the clustering seems correct. In practice the results are much more consistent over several runs for the job running for a day on 32 nodes than for the job running on 189 nodes. This may be explained by the weights found by our method shown above in Figure 4.11. The weights' series are similar for the first but not the second : the period of core computation of the former seems to be the same for all nodes while it is not the case for the latter.

The other issue is that the groups are very imbalanced in the case of the job using 189 nodes : one of the group contains 179 nodes, the others 8 and 2. The groups of the job using 32 nodes are much more balanced, the main one has 22 nodes and the other 10. All the log-returns series share the same variance during core computation, thus for the main group of 179 nodes the parameter β should be 20 times higher than what it should be for small ones. This makes β impossible to tune according to our heuristic since it is shared for all clusters. This may explain why the weights' series of the groups have very different regularity in the jobs running on 189

nodes, the main group has the most piecewise-constant weights' series while the other are much more irregular. It is even clear that the time series of one small group cannot be segmented in core computation phases correctly just by checking that a weight is zero valued, even a threshold could be incorrect.

Handling the fact that groups can be a lot imbalanced remains an issue to solve to have a reliable subspace clustering for our data.

Toward a model for imbalanced clusters

Most of our focus was what penalization on the weights' series should be used. This choice of penalization could be linked with a choice of a prior on the shape of the W of our model (4.1) in a bayesian framework.

However, we also did not look at a prior on the number of nodes in each group. In all the algorithms we used, the number of nodes in each group is expected to be roughly the same. Given a number of groups, this prior is often uniform by default. But it seems that groups of nodes can be very imbalanced in a production job. This may explain the poor results of our model on some production job. The consequence of the unbalancing of the groups may be explained by our heuristic that the ideal β parameter value is proportional to the number of groups but the value β must be shared by all the groups.

It suggests that instead of considering the weighted sum of the squared errors in our objective function, we should consider the weighted average of squared errors. Surprisingly, we found no study on a clustering algorithm based on the minimization of the sum of averages of squared distances to centroids while the best known clustering algorithm, K-means, is based on the minimization of the sum of squared distances to centroids. The explication could be that such algorithm cannot be approximately solved in batches : there is no proof that the way we solve K-means can still be applied to find a local minimum of the sum of the average of squared distances to centroids. This means the resolution must iterate samples by samples and do several rounds before reaching convergence. The properties of such clustering could be also very unusual, some groups may merge together to benefit from an average computed on more samples for example. Such shrinkage effect of groups during the optimization was already described by [Hu et al., 2018] on another clustering algorithm close to K-means.

Such clustering model could be the topic of further studies along with a more formal interpretation on how the hyperparameters of our model should be tuned with such model.

4.4 . Conclusion on the time series from raw bytes' communications

We explore a new type of time series available to better understand the computing center usage and find feature that may be interpreted with expert knowledge. We normalized the data to be able to compare series. Then we model their feature and find that a subspace clustering seems to be a good way to extract them. We propose a model for the subspace clustering designed for time series based on features selection and smoothing penalization. We show that we can use a fast convex optimization solvers to fit the model for a whole class of penalization functions we called gradient-based penalization and we compare the result with the two most representative examples on a small simulated dataset. Finally, we identify a major issue with the most appropriate gradient-based penalization for our data : it extracts only one time period used

for the clustering. We change the penalization for one that is not gradient-based anymore, and propose a new solver for the new penalization that is even faster by removing the need of iterations in the optimization of weights. We checked that several time periods can be reliably selected and looked at the result on real job data.

This work can be extended in many ways. From the side of HPC expertise, we only looked at incoming bytes' communication but many other metrics are collected on computing nodes. Experiments must also be performed to confirm that reliable interpretation can be made from the incoming bytes data aggregated on the node level, in particular what we interpreted as checkpoint restart or steps. These experiments could be used to also evaluate our models. This subject is also a source of original challenges for the machine learning research. The model must be improved to handle highly imbalanced data, the heuristics to set the model hyperparameters also suggest new generic models that we have never seen to the best of our knowledge and deserve more study of their properties.

Joint works, discussions and interactions between statisticians and the operators of the computing center are essential to find new way to manage an HPC center and move forward on the topic of the analysis of industrial count time series.

Conclusion

This thesis manuscript highlights several statistical analysis contributions of the different signals recorded from large-scale computers. The main objective is to optimize the performance of the whole computer center through a better organization of the production or to increase the availability of the computer centers by detecting failures preventively to optimize maintenance. In order to answer this question, we first had to identify use cases in which it is possible to use these techniques during production and then test them.

We first present a simplified model of the computer to understand the interactions between its different parts, whether human or hardware. We detailed the different components and their roles. Our study also looks at the expected behavior of human individuals. We identify the different factors that can influence them and show that the pricing policy of the computer center is already one of the most effective tools available to computer centers to modulate production. This simplified representation reveals which data is accessible to the administration of the computer but also how reliable it is, i.e. whether it can be manipulated by the users. This also allows us to formalize three research tracks where it could be used with statistical learning methods to better understand the use of the calculator or to encourage users to adapt their production to the context. First a prediction of the electricity consumption of a job seems necessary to encourage users to have a less energy-consuming production. Then data visualization tools for large counts could greatly assist in the analysis of system logs, which are generally the only data detailing the activity of the computer but require a lot of time and expertise to be exploited. Finally, the probe data collected by the CEA could be used to classify jobs reliably. We then exploited these three research tracks.

We begin by proposing a simple model that is easy to integrate into production and that predicts the power consumption of a resource allocation to a user according to his request before the SLURM scheduler plans the allocation. This model uses historical data from the SLURM scheduler combined with the energy consumption of each job to make its prediction. We show that this model is able to predict the global consumption of the computing center for a given scheduling, thus opening the possibility to control its global consumption via the scheduling of jobs. This model does not require regular training and is easy to interpret due to its simplicity. Its main drawback is that it is memory-based, so it is not possible to say how much memory is needed. This first success published in the ISC-HPC 2020 conference therefore shows a concrete contribution of statistical methods to the control of a computer.

We then show that it is very difficult to exploit syslog system logs with statistical methods without adding message identifiers as soon as they are generated, because they present too many ambiguities for statistical methods. We therefore focus on logs from the HPSS software and the SeLfiE profiler whose generation allows this exploitation. Their aggregation results in count data with properties that are not often treated together in the literature : high values with scale independence often better treated by compositional data analysis but many structural zeros often better treated by counting models. After normalizing the different quantities and a logarithmic transformation by imputing a mean to the zeros, we use a principal component projection to visualize the low-dimensional data and preserve the possible structures of interest. The result seems more interesting on SeLfiE data than on HPSS data, showing a structural difference bet-

ween these two types of data and the difficulty to treat logs as simple identified messages. The problem thus remains open and needs more attention by experts to be solved.

Finally, we wonder whether it is possible to classify jobs from the time series extracted by the sensors on the computing nodes collected by the CEA. We are first interested in the data generated by benchmarks regularly executed to verify the good performance of the computer during maintenance. We are able to note important differences between certain benchmarks and similarities when expected. We therefore observed data from production jobs to try to find properties close to those observed on the benchmarks. We interpret a production job as a mixture of data generated by the execution of a benchmark on several groups of nodes and over several time periods. We therefore propose to perform this decomposition automatically in order to allow the construction of reliable estimators for job classification. We notice that the decomposition of a job into groups of nodes and computational phases corresponds well to the subspace clustering problem in the machine learning literature. We therefore propose an acceleration of the resolution of one of these problems using convex optimization. This acceleration allows us to propose variants that seem more appropriate and finally to retain a variant where the resolution is more direct, which allows us to scale up the algorithms. The results are promising, especially for the phase segmentation of time series, but the clustering of nodes seems problematic when the number of nodes per cluster is very unbalanced. This is a very common problem for the most used clustering algorithms in machine learning based on the minimization of a power sum of distances. The interpretation of the parameters of our model suggests an original modification of this type of algorithm which should be the subject of future studies in machine learning.

This study is therefore an exploration of the application of machine learning to utilization and monitoring data of computing centers, following the systematic extraction of data from their use setup by HPC experts. It covers a wide range of topics for machine learning such as time series analysis, convex optimization, NLP, online estimator computation, compositional data analysis. It also covers a wide range of topics for HPC such as the structure of the computer, monitoring and resources management software, and communication patterns between computing nodes. It highlights the existence of problems of varying complexity. Some problems could be solved very quickly as soon as the data was made available. Others require more pre-processing by HPC expertise before machine learning can be used. Conversely, other problems require original research in machine learning. This study should be seen as a response of machine learning to the different problems and data presented by HPC experts. It is part of a discussion that needs to be pursued with more interaction between these two fields. It illustrates the need to integrate machine learning expertise with teams of HPC experts to provide concrete tools to help monitor and control large-scale computers based on statistical analysis. This is a critical topic for the race in computing power given the growing constraint on energy production.

A - Duality and convex optimization

A.1 . Fenchel duality

Duality is used to name a method that solves a problem in the dual space E^* of the vector space E of the initial problem that we will suppose euclidean. Fenchel-Rockafeller duality is a theory which allows to apply the duality principles to any problem involving convex and lower semi continuous functions. We show how the definition of the convex conjugate naturally arise from the dual representation of a set and what optimization algorithms are good candidate to solve our problem.

The dual space E^* is the space of linear form of a vector space E. We assume E is euclidean. A linear form in E^* is a scalar product $\langle .,. \rangle$ with a vector y of E and each can be associated with a family of sets in E called half-spaces $\mathcal{H}(y,t)$ defined as

$$\mathcal{H}(y,t) = \{\langle y, . \rangle \leqslant t\} \subset E$$

Any set $C \subset E$ can be mapped to a set $C_* \subset E \times \mathbb{R}$ we named "dual set" of half-spaces parameters $(y,t) \in E \times \mathbb{R}$ which contain C, more formally

$$(y,t) \in C_* \Leftrightarrow (\forall c \in C, \langle y, c \rangle \leqslant t) \Leftrightarrow C \subset \mathcal{H}(y,t)$$

By definition, the dual set C_* of C is the polar cone of the set $(C,-1) \subset E \times \mathbb{R}$. A dual set C_* is also the epigraph of $y \in E \mapsto \sup\{t \in \mathbb{R} | \forall c \in C, \langle y, c \rangle \leqslant t\}$ as a direct consequence of its definition. This function uniquely defines C_* and is convex, lower-semi-continuous and positively homogeneous since its epigraph C_* is a polar cone so a closed convex cone.

Fenchel introduces the concept of convex conjugate function or dual function f^* of a real-valued function f to solve complex convex problems. We allow f to take infinity value, its domain noted $\mathrm{dom}(f)$ is where $f < +\infty$. f is said to be proper if $f > -\infty$ everywhere and $\mathrm{dom}(f) \neq \emptyset$. The common definition of f^* used and introduced by Fenchel is

$$\begin{array}{rccc} f^* : & E & \longrightarrow & \bar{\mathbb{R}} \\ & y & \mapsto & \sup_{x \in E} \langle y, x \rangle - f(x) \end{array}$$

The epigraph of the dual function f^* is related to the dual set $\mathrm{epi}(f)_* \subset (E \times \mathbb{R}) \times \mathbb{R}$ of the epigraph of f as using the definition of f^* we have

$$(y,s) \in \mathrm{epi}(f^*) \Leftrightarrow \forall (x,t) \in \mathrm{epi}(f), \langle y, x \rangle + (-1) \times t = \langle (y,-1), (x,t) \rangle \leqslant s \Leftrightarrow ((y,-1),s) \in \mathrm{epi}(f)_*$$

This means the epigraph of f^* is the restriction of $\mathrm{epi}(f)_*$ to the affine subspace $(E,-1,\mathbb{R})$ or the intersection of the polar cone of $(\mathrm{epi}(f),-1) \subset E \times \mathbb{R} \times \mathbb{R}$ with $(E,-1,\mathbb{R})$ that we can also write $\{(y,-1,s) | (y,s) \in \mathrm{epi}(f^*)\}$. This implies that $\mathrm{epi}(f^*)$ is a closed convex set proving that f^* is convex and lower-semi-continuous. If f is a proper function, we can show that the cone generated by $\{(y,-1,s) | (y,s) \in \mathrm{epi}(f^*)\}$ is indeed the polar cone of $(\mathrm{epi}(f),-1) \subset E \times \mathbb{R} \times \mathbb{R}$ the same way as [Rockafellar, 1970, Theorem 14.4 on p.124].

The dual function is useful to solve complex problems thanks to the Fenchel-Moreau theorem which states when the dual operation is an involution. Using f^* in place of f in the above property

and exchanging the last two coordinates, we obtain that the cone generated by $(\mathrm{epi}(f^{**}), -1) \subset E \times \mathbb{R} \times \mathbb{R}$ is the polar cone of $\{(y, -1, s) | (y, s) \in \mathrm{epi}(f^*)\}$. We already said that the former generated cone is also the polar cone of $(\mathrm{epi}(f), -1)$. This means that the cone generated by $(\mathrm{epi}(f^{**}), -1)$ is the polar cone of the polar cone of $(\mathrm{epi}(f), -1)$. The polar cone theorem (deduced from Hahn-Banach theorem or axiom of choice) states that the polar cone of the polar cone of a set is the smallest closed convex cone which contains this set. It implies that $\mathrm{epi}(f^{**})$ is the smallest closed convex set containing $\mathrm{epi}(f)$. The biconjugate f^{**} is the highest convex and lower-semi-continuous function below f. If f is already convex and lower-semi-continuous, then f^{**} is f because $\mathrm{epi}(f)$ is a closed convex set so $\mathrm{epi}(f^{**}) = \mathrm{epi}(f)$. The convex conjugation is a one-to-one mapping for convex lower-semi-continuous functions and is its own inverse.

The highest lower-semi-continuous function below f must match f where it is lower-semi-continuous as it is everywhere equal to the lower limit of f at any point. If f is convex then it is also convex because its epigraph is the closure of the convex set $\mathrm{epi}(f)$. It implies that for f convex, $f^{**}(z) = f(z)$ if and only if f is lower-semi continuous at point z. This Fenchel-Moreau equality can be interpreted as a valid case of min-max inversion :

$$\begin{aligned} f^{**}(z) = &\quad \sup_{y \in E} \inf_{x \in E} \quad \langle y, z - x \rangle + f(x) \\ f(z) = &\quad \inf_{x \in E} \sup_{y \in E} \quad \langle y, z - x \rangle + f(x) \end{aligned}$$

A.2 . Convex optimization with duality

To solve an optimization problem, called primal problem in the context of convex optimization, a duality method uses a smoother convex function F, also called the perturbation function, of two variables such that the function of the first variable matches the objective function when the second variable (interpreted as a perturbation) is 0. The value of the optimization problem, or primal value, is then the value at 0 of the minimum of the perturbation function F as a function of the second variable, also called the value function ϑ, which is known to be convex as soon as F is convex.

The value $\vartheta^{**}(0)$ of the biconjugate of the value function at 0 is called the dual value. We can show it is by definition the supremum of $-F^*$ as a function of the perturbation function when its first variable is set to 0. This maximization problem is called the dual problem and we know from the inequality $\vartheta^{**}(0) \leqslant \vartheta(0)$ that the value of the dual problem is then less than the primal value. We say weak duality holds and the difference between the primal value and the dual value is called the duality gap. When the value function ϑ is lower-semi-continuous at 0 then $\vartheta^{**}(0) = \vartheta(0)$ by the Fenchel-Moreau equality, this means the value of the primal convex minimization problem is equal to the one of a dual concave maximization deduced from the dual of the perturbation. In such cases where primal and dual value are equal, meaning the duality gap is zero, we say that strong duality holds.

The Fenchel-Rockafellar duality corresponds to a minimization problem of the form (4.7) where f, g can be any proper convex lower-semi-continuous functions of a real vector space.

$$\inf_{X \in \mathbb{R}^T} f(KX) + g(X) \tag{4.7}$$

An interesting perturbation function, such that $F(X, 0) = f(KX) + g(X)$, is

$$\begin{array}{rccc} F: & \mathbb{R}^T \times \mathbb{R}^{T-1} & \longrightarrow & \mathbb{R} \\ & (X,Y) & \mapsto & f(KX+Y)+g(X) \end{array}$$

The value function is $\vartheta : Y \in \mathbb{R}^{T-1} \mapsto \inf_{X\in\mathbb{R}^T} F(X,Y)$ for which we have

$$\vartheta(0) = \inf_{X\in\mathbb{R}^T} f(KX)+g(X)$$

Its biconjugate at 0 is then $\sup_{Y\in\mathbb{R}^{T-1}} -F^*(0,Y)$
It is the associated dual problem that we deduce using the expression of F^*

$$\begin{aligned} \vartheta^{**}(0) &= \sup_{Y\in\mathbb{R}^{T-1}} -F^*(0,Y) \\ &= \sup_{Y\in\mathbb{R}^{T-1}} - \sup_{(X',Y')\in\mathbb{R}^T\times\mathbb{R}^{T-1}} \langle Y,Y'\rangle - F(X',Y') \\ &= \sup_{Y\in\mathbb{R}^{T-1}} - \sup_{(X',Y')\in\mathbb{R}^T\times\mathbb{R}^{T-1}} \langle Y,Y'\rangle - f(KX'+Y') - g(X') \\ &= \sup_{Y\in\mathbb{R}^{T-1}} - \sup_{X'\in\mathbb{R}^T} f^*(Y) - \langle KX',Y\rangle - g(X') \\ &= \sup_{Y\in\mathbb{R}^{T-1}} -f^*(Y) - (\sup_{X'\in\mathbb{R}^T} \langle X',-K^*Y\rangle - g(X')) \\ &= \sup_{Y\in\mathbb{R}^{T-1}} -g^*(-K^*Y) - f^*(Y) \end{aligned} \tag{A.1}$$

Under sufficient conditions, the value function is convex and lower-semi-continuous at 0. Using Fenchel-Moreau equality we conclude that strong duality holds. The sufficient conditions are often related to the existence of a point in the relative interior of domain of the involved functions. The relative interior of a set is its interior for the topology of the smallest affine set containing it, it is known that a convex function is continuous on the relative interior of its domain. The sufficient conditions formulated by [Rockafellar, 1970, Corollary 31.2.1 on p.327] are the existence of $X \in \mathbb{R}^T$ in the relative interior of $\mathrm{dom}(g)$ such that KX is in the relative interior of $\mathrm{dom}(f)$ or the existence of $Y \in \mathbb{R}^{T-1}$ in the relative interior of $\mathrm{dom}(f^*)$ such that $-K^*Y$ is in the relative interior of $\mathrm{dom}(g^*)$. These two conditions are called respectively primal and dual strict feasibility conditions and points that verify them are called strict feasible points because they are values for which the objective function of the primal or dual problems is not infinite. The existence of primal and dual strict feasible points respectively also imply that the maximum of the dual problem respectively the minimum of the primal problem is attained.

Using that $f = f^{**}$ in (4.7), we can write a primal dual formulation of our problem as finding a saddle point of a saddle function as (A.2). If the two feasibility conditions are satisfied then the minimum and supremum are attained and the minimum and maximum can be exchanged and there is a saddle-point whose coordinates are the solutions of the primal and dual problems. When fulfilled, these conditions allows using very efficient optimization algorithm.

$$\min_{X\in\mathbb{R}^T} \max_{Y\in\mathbb{R}^{T-1}} \langle KX,Y\rangle + g(X) - f^*(Y) \tag{A.2}$$

An illustration of the use of a saddle-point problem is shown Figure A.1 on a toy example. The primal problem is to minimize $h_1(x) = 0.5x^2 + \delta_{|.|\leqslant 1}(x+2)$ which corresponds to finding the minimum of $x \mapsto 0.5x^2$ on $[-3,-1]$. The surface is a plot of $h(x,y) = 0.5x^2 - |y| + (x+2)y$

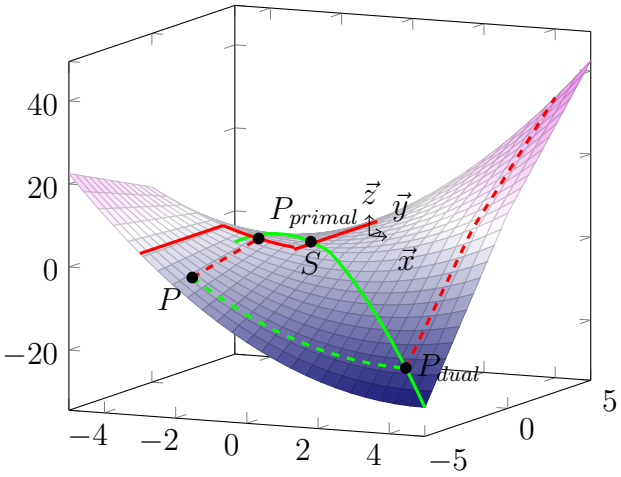


Figure A.1 – Illustration of saddle-point duality with a toy problem. The surface is a plot of a saddle function that can be used to minimize the square function on the interval $[-3, -1]$.

which is a saddle function so that the dual problem is minimizing $h_2(y) = 2y - |y| - 0.5y^2$. For any feasible point $P = (x, y)$, we construct (dashed lines) $P_{primal} = (x, y_1)$ in red and $P_{dual} = (x_2, y)$ in green where $y_1 \in \operatorname{argmax}_y h(x, y)$ (almost always 0 or infinite-valued, any positive or negative respectively values are possible if x is respectively -3 or -1) and $x_2 = \operatorname{argmin}_x h(x, y) = -y$ so that $h(P_{primal}) = h_1(x)$ and $h(P_{dual}) = h_2(y)$. They are the values of the primal and dual problems when x and y are set to 0. By construction $h(P_{primal}) \geq h(P) \geq h(P_{dual})$, this is the weak duality property : the set of values of the primal problem (shown in plain red) is above the one of dual problem (shown in plain green), this could be written $\inf_P h(P_{primal}) \geq \sup_P h(P_{dual})$. Strong duality is when these two sets are both arbitrarily close to the same height. It is the case as the two sets intersect at a saddle point $S = (-1, 1)$ which match the trivial primal solution $x = -1$. P_{dual} is also an example of a point that is only dual feasible and not primal feasible since its first coordinate does not satisfy the constraints of the primal problem, this can be seen by the red dashed lines going to infinity.

A.3 . Chambolles-Pock algorithm

Assuming strong duality hold, we can use a primal-dual method to solve our problem. A primal-dual algorithm alternates between solving the primal problem (4.7) and the dual problem (A.1) at each iteration. The iteration can be visualized on Figure A.1. From the current solution (x, y) on the algorithm fixes y and makes a step toward the minimum in the primal space (in red) to a new x to minimize $h(P_{primal})$ then fixes y and makes a step toward the maximum in the dual space (in green) to a new y to maximize $h(P_{dual})$ and repeat until convergence.

When dealing with known convex functions, it is often more practical to apply the proximal operator instead of doing a gradient step if it can be computed analytically. The definition of proximal operator is

$$\underset{h}{\operatorname{prox}}(X) = \underset{Y}{\operatorname{argmin}}\, h(Y) + \frac{1}{2}||X - Y||_2^2$$

prox_f can be found using sub-gradient optimal conditions. The proximal operator of the cha-

racteristic function of a convex set is the projection on it, a vector is mapped to the closest one in this set.

[Chambolle and Pock, 2011] proposed a primal-dual algorithm to solve the problem (A.2) when the proximal operators of f and g are known and the problem has a saddle-point, which is our case. The algorithm of [Chambolle and Pock, 2011] is given in pseudocode 3. It alternates proximal gradient descent with an equivalent of step size σ for the primal problem (4.7) and proximal gradient ascent of step size τ for the dual problem (A.1). At the end of an iteration, the current primal solution is updated by extrapolation with the new primal solution. The parameter θ tunes this extrapolation and is fixed to 1 in the algorithm of [Chambolle and Pock, 2011]. Previous primal-dual algorithm also have this form, like Arrow-Hurwicz studied in [Zhu and Chan, 2008] but there is no extrapolation, θ is 0. This parameter can also be tuned differently if the involved functions are strictly convex to get convergence rate to the primal solution. The primal-dual algorithm of [Chambolle and Pock, 2011] is the first with a proof of convergence and assumption only on the step size of the two proximal methods. The step parameters σ and τ must be chosen such that $\sigma\tau||K||^2 < 1$ to ensure that the algorithm converges in the case $\theta = 1$. [Chambolle and Pock, 2011] also proves that Arrow-Hurwicz converges but only when the feasible set is bounded, which is also our case.

Algorithm 3 Chambolle's algorithm

First-order primal-dual algorithm, $\theta = 1$ is Chambolle's algorithm

Require: prox_{f^*}, prox_g, K and K^*, σ and τ the gradient step parameters and n_{iter} the number of iteration and initial values for the solutions (X^0, Y^0)

Ensure: $\sigma\tau||K||^2 < 1$

$\theta \leftarrow 1$
$\bar{X} \leftarrow X^0$
$X \leftarrow X^0$
$Y \leftarrow Y^0$
for $n = 1 \dots n_{iter}$ **do**
  $X_{old} = X$
  $Y = \text{prox}_{\sigma f^*}(Y + \sigma K\bar{X})$
  $X = \text{prox}_{\tau g}(X - \tau K^* Y)$
  $\bar{X} = X + \theta(X - X_{old})$
end for
return (X, Y)

The only constraint on σ and τ is that $\sigma\tau||K||^2$ is smaller than 1. Making this product as close as possible (but not equal) to 1 minimize the number of iterations required to be close to the solution. However, this only constraints the product of σ and τ and minimizing the number of iterations further may require different step size in primal and dual space depending on the problem. To keep the number of iterations the same for many problems of the same form, we can use the linesearch method proposed by [Malitsky and Pock, 2016]. The parameters σ and τ are adjusted at each iteration to have a ratio of β until a stopping condition is reached as shown in Algorithm 4.

The linesearch parameters $\delta, \mu \in]0, 1[$ respectively tune the largest primal step size we accept at each epoch to avoid that $\sigma\tau||K||^2 \geqslant 1$ and how much the primal step size is reduced if it is too big. The resulting choice of step size is theoretically better if these two parameters are close to 1

(but not 1). In practice, the linesearch is longer if μ is closer to 1 and limit the speedup provided by choosing δ too close to 1.

Algorithm 4 Primal-Dual proximal splitting with linesearch [Malitsky and Pock, 2016]

Require: prox_{f^*}, prox_g, K and K^*, $\tau_0 > 0$ the initial primal gradient step size parameter, β the ratio of primal and dual gradient step size, n_{iter} the number of iterations, μ and δ the precision of linesearch estimation and initial values for the solutions (X^0, Y^0)

$\theta \leftarrow 1$

$\bar{x} \leftarrow x$

for $k \in 1, ..., n\_iter$ **do**

$x_{old} \leftarrow x$

$x \leftarrow \text{prox}_{\tau_{k-1} g}(x - \tau_{k-1} K^* y)$

$\tau_k \leftarrow \tau_{k-1}(1 + \sqrt{\theta_{k-1}})/\mu$

repeat ▷ Linesearch loop

$\tau_k \leftarrow \mu \tau_k$

$\theta_k \leftarrow \frac{\tau_k}{\tau_{k-1}}$

$\bar{x} \leftarrow x + \theta_k (x - x_{old})$

$y^{k+1} \leftarrow \text{prox}_{\beta \tau_k f^*}(y^k + \beta \tau_k K \bar{x})$

until $\sqrt{\beta} \tau_k ||K^* y^{k+1} - K^* y^k|| \leqslant \delta ||y^{k+1} - y^k||$ ▷ Stopping condition

end for

return (x, y) ▷ (x, y) primal-dual solutions of A.2

Bibliographie

[Aitchison, 1982] Aitchison, J. (1982). The statistical analysis of compositional data. *Journal of the Royal Statistical Society : Series B (Methodological)*, 44(2) :139–160.

[Aussel et al., 2018] Aussel, N., Petetin, Y., and Chabridon, S. (2018). Improving performances of log mining for anomaly prediction through nlp-based log parsing. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 237–243. IEEE.

[AWS, 2022] AWS (2022). Amazon ec2 pricing. Last seen 03/2022 https://aws.amazon.com/ec2/pricing/?nc1=h_ls.

[Bellego et al., 2021] Bellego, C., Benatia, D., and Pape, L.-D. (2021). Dealing with logs and zeros in regression models. *CREST-Série des Documents de Travail*, (2019-13).

[Borghesi et al., 2016] Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., and Benini, L. (2016). Predictive modeling for job power consumption in hpc systems. In *International conference on high performance computing*, pages 181–199. Springer.

[Borghesi et al., 2018] Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., and Benini, L. (2018). Scheduling-based power capping in high performance computing systems. *Sustainable Computing : Informatics and Systems*, 19 :1–13.

[Bugbee et al., 2017] Bugbee, B., Phillips, C., Egan, H., Elmore, R., Gruchalla, K., and Purkayastha, A. (2017). Prediction and characterization of application power use in a high-performance computing environment. *Statistical Analysis and Data Mining : The ASA Data Science Journal*, 10(3) :155–165.

[Burbidge et al., 1988] Burbidge, J. B., Magee, L., and Robb, A. L. (1988). Alternative transformations to handle extreme values of the dependent variable. *Journal of the American Statistical Association*, 83(401) :123–127.

[B.V., 2022] B.V., E. (2022). Logstash's software repository. GitHub repository : https://github.com/elastic/logstash.

[Calder et al., 2010] Calder, J., Mansouri, A., and Yezzi, A. (2010). Image sharpening via sobolev gradient flows. *SIAM Journal on Imaging Sciences*, 3(4) :981–1014.

[Carasso, 2007] Carasso, D. (2007). Semi-automatic discovery of extraction patterns for log analysis.

[CEA, 2022a] CEA (2022a). Hardware specification of cobalt computing center. Last seen 06/2022 http://www-ccrt.cea.fr/fr/moyen_de_calcul/cobalt.htm (In french).

[CEA, 2022b] CEA (2022b). Hardware specification of irene computing center. Last seen 06/2022 http://www-hpc.cea.fr/en/complexe/tgcc-JoliotCurie.htm.

[Chambolle and Pock, 2011] Chambolle, A. and Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1) :120–145.

[Condat, 2013] Condat, L. (2013). A direct algorithm for 1-d total variation denoising. *IEEE Signal Processing Letters*, 20(11) :1054–1057.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1) :21–27.

[Dagum and Menon, 1998] Dagum, L. and Menon, R. (1998). Openmp : an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1) :46–55.

[Davies and Kovac, 2001] Davies, P. L. and Kovac, A. (2001). Local extremes, runs, strings and multiresolution. *The Annals of Statistics*, 29(1) :1–65.

[Donders et al., 2006] Donders, A. R. T., Van Der Heijden, G. J., Stijnen, T., and Moons, K. G. (2006). A gentle introduction to imputation of missing values. *Journal of clinical epidemiology*, 59(10) :1087–1091.

[Du and Li, 2016] Du, M. and Li, F. (2016). Spell : Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE.

[Dubois et al., 2014] Dubois, M., Hadj-Selem, F., Löfstedt, T., Perrot, M., Fischer, C., Frouin, V., and Duchesnay, E. (2014). Predictive support recovery with tv-elastic net penalty and logistic regression : an application to structural mri. In *Pattern Recognition in Neuroimaging, 2014 International Workshop on*, pages 1–4. IEEE.

[Dutot et al., 2017] Dutot, P.-F., Georgiou, Y., Glesser, D., Lefevre, L., Poquet, M., and Rais, I. (2017). Towards energy budget control in hpc. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press.

[Eckner,] Eckner, A. Algorithms for unevenly-spaced time series : Moving averages and other rolling operators.

[Etienne and Latifa, 2014] Etienne, C. and Latifa, O. (2014). Model-based count series clustering for bike sharing system usage mining : A case study with the vélib'system of paris. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3) :39.

[Evans and Gariepy, 1991] Evans, L. C. and Gariepy, R. F. (1991). *Measure Theory and Fine Properties of Functions*, volume 5. CRC Press.

[Gabriel et al., 2004] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI : Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary.

[Gama et al., 2014] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4) :1–37.

[Gaussier et al., 2015] Gaussier, E., Glesser, D., Reis, V., and Trystram, D. (2015). Improving backfilling by using machine learning to predict running times. In *SC'15 Proceedings*, pages 1–10. IEEE.

[Georgiou et al., 2014] Georgiou, Y., Cadeau, T., Glesser, D., Auble, D., Jette, M., and Hautreux, M. (2014). Energy accounting and control with slurm resource and job management system. In *International Conference on Distributed Computing and Networking*, pages 96–118. Springer.

[Ghiasvand and Ciorba, 2017] Ghiasvand, S. and Ciorba, F. M. (2017). Event pattern identification in anonymized system logs.

[Ghiasvand et al., 2016] Ghiasvand, S., Ciorba, F. M., and Nagel, W. E. (2016). Turning privacy constraints into syslog analysis advantage. 29th ACM/IEEE International Conference for High Performance Computing ….

[Gonze et al., 2020] Gonze, X., Amadon, B., Antonius, G., Arnardi, F., Baguet, L., Beuken, J.-M., Bieder, J., Bottin, F., Bouchet, J., Bousquet, E., et al. (2020). The abinit project : Impact, environment and recent developments. *Computer Physics Communications*, 248 :107042.

[Gupta et al., 2013] Gupta, A., Kale, L. V., Gioachin, F., March, V., Suen, C. H., Lee, B.-S., Faraboschi, P., Kaufmann, R., and Milojicic, D. (2013). The who, what, why, and how of high performance computing in the cloud. In *2013 IEEE 5th international conference on cloud computing technology and science*, volume 1, pages 306–314. IEEE.

[Harms and Yamartino, 2010] Harms, R. and Yamartino, M. (2010). The economics of the cloud. *Microsoft whitepaper, Microsoft Corporation*, 3 :157.

[He et al., 2016] He, P., Zhu, J., He, S., Li, J., and Lyu, M. R. (2016). An evaluation study on log parsing and its use in log mining. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 654–661. IEEE.

[He et al., 2017] He, P., Zhu, J., Zheng, Z., and Lyu, M. R. (2017). Drain : An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE.

[Heroux et al., 2013] Heroux, M. A., Dongarra, J., and Luszczek, P. (2013). Hpcg benchmark technical specification. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[Holt, 2004] Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1) :5–10.

[hpss collaboration.org, 2021] hpss collaboration.org (2021). Hpss error manual. can be found at https://www.hpss-collaboration.org/documents/HPSS_9.2.0_Error_Manual.pdf.

[Hron et al., 2010] Hron, K., Templ, M., and Filzmoser, P. (2010). Imputation of missing values for compositional data using classical and robust methods. *Computational Statistics & Data Analysis*, 54(12) :3095–3107.

[Hu et al., 2020] Hu, C., Nakano, M., and Okumura, M. (2020). Autoencoder guided bootstrapping of semantic lexicon. *Journal of Natural Language Processing*, 27(3) :627–652.

[Hu et al., 2018] Hu, C. W., Li, H., and Qutub, A. A. (2018). Shrinkage clustering : a fast and size-constrained clustering algorithm for biomedical applications. *BMC bioinformatics*, 19(1) :1–11.

[Huang et al., 2005] Huang, J. Z., Ng, M. K., Rong, H., and Li, Z. (2005). Automated variable weighting in k-means type clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5) :657–668.

[Huang et al., 2016] Huang, X., Ye, Y., Xiong, L., Lau, R. Y., Jiang, N., and Wang, S. (2016). Time series k-means : A new k-means type smooth subspace clustering for time series data. *Information Sciences*, 367 :1–13.

[Hunter, 1986] Hunter, J. S. (1986). The exponentially weighted moving average. *Journal of quality technology*, 18(4) :203–210.

[Johnson, 1949] Johnson, N. L. (1949). Systems of frequency curves generated by methods of translation. *Biometrika*, 36(1/2) :149–176.

[Karlin et al., 2013] Karlin, I., Keasler, J., and Neely, J. (2013). Lulesh 2.0 updates and changes. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[Klinkenberg et al., 2017] Klinkenberg, J., Terboven, C., Lankes, S., and Müller, M. S. (2017). Data mining-based analysis of hpc center operations. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 766–773. IEEE.

[Knüpfer et al., 2008] Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S., and Nagel, W. E. (2008). The vampir performance analysis tool-set. In *Tools for high performance computing*, pages 139–155. Springer.

[Laurent Nguyen, 2017] Laurent Nguyen, Dominique Martinet, A. C. (2017). Self and light profiling engine - a very light profiling tools for linux commands and hpc codes. GitHub repository : https://github.com/cea-hpc/selFIe.

[Le and Cuturi, 2013] Le, T. and Cuturi, M. (2013). Generalized aitchison embeddings for histograms. In *Asian conference on machine learning*, pages 293–308. PMLR.

[Ledoit and Wolf, 2004] Ledoit, O. and Wolf, M. (2004). Honey, i shrunk the sample covariance matrix. *The Journal of Portfolio Management*, 30(4) :110–119.

[Liao, 2005] Liao, T. W. (2005). Clustering of time series data—a survey. *Pattern recognition*, 38(11) :1857–1874.

[Little and Rubin, 2019] Little, R. J. and Rubin, D. B. (2019). *Statistical analysis with missing data*, volume 793. John Wiley & Sons.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2) :129–137.

[Malitsky and Pock, 2016] Malitsky, Y. and Pock, T. (2016). A first-order primal-dual algorithm with linesearch. *arXiv preprint arXiv :1608.08883*.

[Manning et al., 2008] Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

[Mitchell, 1980] Mitchell, T. M. (1980). *The need for biases in learning generalizations*.

[Möller-Levet et al., 2003] Möller-Levet, C. S., Klawonn, F., Cho, K.-H., and Wolkenhauer, O. (2003). Fuzzy clustering of short time-series and unevenly distributed sampling points. In *International Symposium on Intelligent Data Analysis*, pages 330–340. Springer.

[Mu'alem and Feitelson, 2001] Mu'alem, A. W. and Feitelson, D. G. (2001). Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE transactions on parallel and distributed systems*, 12(6) :529–543.

[Nikitenko et al., 2016] Nikitenko, D., Stefanov, K., Zhumatiy, S., Voevodin, V., Teplov, A., and Shvets, P. (2016). System monitoring-based holistic resource utilization analysis for every user of a large hpc center. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 305–318. Springer.

[O'Hara and Kotze, 2010] O'Hara, R. and Kotze, J. (2010). Do not log-transform count data. *Nature Precedings*, pages 1–1.

[OVHcloud, 2022] OVHcloud (2022). Our dedicated server ranges. Last seen 03/2022 https://us.ovhcloud.com/bare-metal/.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn : Machine learning in python. *Journal of machine learning research*, 12(Oct) :2825–2830.

[Platini, 2020] Platini, M. (2020). *Apprentissage machine appliqué à l'analyse et à la prédiction des défaillances dans les systèmes HPC*. Theses, Université Grenoble Alpes [2020-....].

[Pospieszny, 2012] Pospieszny, M. (2012). Electricity in hpc centres. https://prace-ri.eu/wp-content/uploads/hpc-centre-electricity-whitepaper-2.pdf.

[Rand, 1971] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336) :846–850.

[Riloff et al., 1999] Riloff, E., Jones, R., et al. (1999). Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479.

[Rockafellar, 1970] Rockafellar, R. T. (1970). *Convex analysis*. Princeton university press.

[Russell and , 2010] Russell, S. J. and , P. N. (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

[Saillant et al., 2020] Saillant, T., Weill, J.-C., and Mougeot, M. (2020). Predicting job power consumption based on rjms submission data in hpc systems. In *International Conference on High Performance Computing*, pages 63–82. Springer.

[Salfner et al., 2004] Salfner, F., Tschirpke, S., and Malek, M. (2004). Comprehensive logfiles for autonomic systems. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 211. IEEE.

[Shepler et al., 2003] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. (2003). Rfc3530 : Network file system (nfs) version 4 protocol.

[Shi et al., 2016] Shi, H.-J. M., Tu, S., Xu, Y., and Yin, W. (2016). A primer on coordinate descent algorithms. *arXiv preprint arXiv :1610.00040*.

[Sîrbu and Babaoglu, 2016] Sîrbu, A. and Babaoglu, O. (2016). Power consumption modeling and prediction in a hybrid cpu-gpu-mic supercomputer. In *European Conference on Parallel Processing*, pages 117–130. Springer.

[Sîrbu and Babaoglu, 2018] Sîrbu, A. and Babaoglu, O. (2018). A data-driven approach to modeling power consumption for a hybrid supercomputer. *Concurrency and Computation : Practice and Experience*, 30(9) :e4410.

[Strehl and Ghosh, 2002] Strehl, A. and Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec) :583–617.

[Tanash et al., 2019] Tanash, M., Dunn, B., Andresen, D., Hsu, W., Yang, H., and Okanlawon, A. (2019). Improving hpc system performance by predicting job resources via supervised machine learning. In *Proceedings of the PEARC*, page 69. ACM.

[Templ et al., 2017] Templ, M., Hron, K., and Filzmoser, P. (2017). Exploratory tools for outlier detection in compositional data with structural zeros. *Journal of Applied Statistics*, 44(4) :734–752.

[Terpstra et al., 2010] Terpstra, D., Jagode, H., You, H., and Dongarra, J. (2010). Collecting performance data with papi-c. In *Tools for High Performance Computing 2009*, pages 157–173. Springer.

[Thelen and Riloff, 2002] Thelen, M. and Riloff, E. (2002). A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the 2002 conference on empirical methods in natural language processing (EMNLP 2002)*, pages 214–221.

[Vaarandi and Pihelgas, 2015] Vaarandi, R. and Pihelgas, M. (2015). Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*, pages 1–7. IEEE.

[Vandenberghe, 2010] Vandenberghe, L. (2010). The cvxopt linear and quadratic cone program solvers. *Online : http ://cvxopt. org/documentation/coneprog. pdf*.

[Vetter, 2013] Vetter, J. S. (2013). *Contemporary high performance computing : from Petascale toward exascale*. CRC Press.

[Vidal et al., 2005] Vidal, R., Ma, Y., and Sastry, S. (2005). Generalized principal component analysis (gpca). *IEEE transactions on pattern analysis and machine intelligence*, 27(12) :1945–1959.

[Vulić and Moens, 2013] Vulić, I. and Moens, M. F. (2013). A study on bootstrapping bilingual vector spaces from non-parallel data (and nothing else). In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1613–1624.

[Wang and Carreira-Perpinán, 2013] Wang, W. and Carreira-Perpinán, M. A. (2013). Projection onto the probability simplex : An efficient algorithm with a simple proof, and an application. *arXiv preprint arXiv :1309.1541*.

[Webb et al., 2016] Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., and Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4) :964–994.

[Xu et al., 2009a] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. (2009a). Largescale system problem detection by mining console logs. *Proceedings of SOSP'09*.

[Xu et al., 2009b] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. (2009b). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132.

[Yamamoto et al., 2018] Yamamoto, K., Tsujita, Y., and Uno, A. (2018). Classifying jobs and predicting applications in hpc systems. In Yokota, R., Weiland, M., Keyes, D., and Trinitis, C., editors, *High Performance Computing*, pages 81–99, Cham. Springer International Publishing.

[Yoo et al., 2003] Yoo, A. B., Jette, M. A., and Grondona, M. (2003). Slurm : Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer.

[Zhu et al., 2019] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering : Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE.

[Zhu and Chan, 2008] Zhu, M. and Chan, T. (2008). An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *UCLA CAM Report*, pages 08–34.

[Zou and Hastie, 2005] Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the royal statistical society : series B (statistical methodology)*, 67(2) :301–320.

[Zumbach and Müller, 2001] Zumbach, G. and Müller, U. (2001). Operators on inhomogeneous time series. *International Journal of Theoretical and Applied Finance*, 4(01) :147–177.