Apprentissage machine appliqué à l'analyse et à la prédiction des défaillances dans les systèmes HPC

Marc Platini

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L' UNIVERSITE GRENOBLE ALPES

Spécialité :Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Marc PLATINI

Thèse dirigée par **Noel DE PALMA**
Et coencadrée par **Thomas ROPARS**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Apprentissage machine appliqué à l'analyse et à la prédiction des défaillances dans les systèmes HPC

Thèse soutenue publiquement le **20 mai 2020**,
devant le jury composé de :

Franck CAPPELLO
Directeur de recherche, Argonne National Laboratory, Rapporteur
Jean-Marc MENAUD
Professeur, IMT-Atlantique, Rapporteur
Sara BOUCHENAK
Professeur, INSA Lyon, Examinatrice, Présidente
Benoit PELLETIER
Directeur de section R&D, ATOS, Examinateur
Noël DE PALMA
Professeur, Université Grenoble Alpes, Directeur de thèse
Thomas ROPARS
Maitre de conférence, Université Grenoble Alpes, Co-Encadrant de thèse

Résumé

Les systèmes informatiques dédiés à la haute performance (HPC) se livrent à une course à la puissance de calcul. Cette course se concrétise principalement par une augmentation de leur taille et de leur complexité. Cependant, cette augmentation entraîne des défaillances fréquentes qui peuvent réduire la disponibilité des systèmes HPC.

Pour gérer ces défaillances et être capable de réduire leur influence sur les systèmes HPC, il est important de mettre en place des solutions permettant de comprendre les défaillances, voire de les prédire. En effet, les systèmes HPC produisent une grande quantité de données de supervision qui contiennent de nombreuses informations utiles à propos de leur état de fonctionnement. Cependant, l'analyse de ces données n'est pas facile à réaliser et peut être très fastidieuse car elles reflètent la complexité et la taille des systèmes HPC. Les travaux présentés dans cette thèse proposent d'utiliser des solutions d'apprentissage machine pour réaliser de manière automatisée cette analyse. De manière plus précise, cette thèse présente deux contributions principales : la première s'intéresse à la prédiction des surchauffes des processeurs dans les systèmes HPC, la deuxième se concentre sur l'analyse et la mise en évidence des relations entre les événements présents dans les journaux systèmes. Ces deux contributions sont évaluées sur des données réelles provenant d'un système HPC de grande taille utilisé en production.

Pour prédire les surchauffes de processeur, nous proposons une solution qui utilise uniquement la température des processeurs. Elle repose sur l'analyse de la forme générale de la température avant un événement de surchauffe et sur l'apprentissage automatisé des corrélations entre cette forme et les événements de surchauffe grâce à un modèle d'apprentissage supervisé. L'utilisation de la forme générale des courbes et d'un modèle d'apprentissage supervisé permet l'apprentissage en utilisant des données de température avec une faible précision et en utilisant un nombre de cas de surchauffe restreint. L'évaluation de la solution montre qu'elle est capable de prédire plusieurs minutes en avance les surchauffes avec une précision et un rappel élevés. De plus, l'évaluation de ces résultats montre qu'il est possible d'utiliser des actions préventives reposant sur les prédictions faites par la solution pour réduire l'influence des surchauffes sur le système.

Pour analyser et mettre en évidence de manière automatisée les relations causales entre dans les événements décrits dans les journaux des systèmes HPC, nous proposons une utilisation détournée d'un modèle d'apprentissage machine profond. En effet, ce type de modèle est classiquement utilisé pour des tâches de prédiction. Grâce à l'ajout d'une nouvelle couche proposée par des travaux de l'état de l'art étudiant les méthodes d'apprentissage machine, il est possible de déterminer l'importance des entrées de l'algorithme dans sa prédiction. En utilisant les informations sur l'importance des entrées, nous sommes capables de reconstruire les relations entre les différents événements. L'évaluation de la solution montre qu'elle est capable de mettre en évidence les relations de la grande majorité des événements survenant sur un système HPC. De plus, son évaluation par des administrateurs montre la validité des corrélations mises en évidence.

Les deux contributions et leurs évaluations montrent le bénéfice de l'utilisation de solutions d'apprentissage machine pour la compréhension et la prédiction des défaillances dans les systèmes HPC en automatisant l'analyse des données de supervision.

Abstract

With the increase in size of supercomputers, also increases the number of failures or abnormal events. This increase of the number of failures reduces the availability of these systems.

To manage these failures and be able to reduce their impact on HPC systems, it is important to implement solutions to understand the failures and to predict them. HPC systems produce a large amount of monitoring data that contains useful information about the status of these systems. However, the analysis of these data is difficult and can be very tedious because these data reflect the complexity and the size of HPC systems. The work presented in this thesis proposes to use machine-learning-based solutions to analyse these data in an automated way. More precisely, this thesis presents two main contributions : the first one focuses on the prediction of processors overheating events in HPC systems, the second one focuses on the analysis and the highlighting of the relationships between the events present in the system logs. Both contributions are evaluated on real data from a large HPC system used in production.

To predict CPU overheating events, we propose a solution that uses only the temperature of the CPUs. It is based on the analysis of the general shape of the temperature prior to an overheating event and on the automated learning of the correlations between this shape and overheating events using a supervised learning model. The use of the general curve shape and a supervised learning model allows learning using temperature data with low accuracy and using a limited number of overheating events. The evaluation of the solution shows that it is able to predict overheating events several minutes in advance with high accuracy and recall. Furthermore, the evaluation of these results shows that it is possible to use preventive actions based on the predictions made by the solution to reduce the impact of overheating events on the system.

To analyze and to extract in an automated way the causal relations between the events described in the HPC system logs, we propose an unconventional use of a deep machine learning model. Indeed, this type of model is classically used for prediction tasks. Thanks to the addition of a new layer proposed by state-of-the-art contributions of the machine learning community, it is possible to determine the weight of the algorithm inputs associated to its prediction. Using this information, we are able to detect the causal relations between the different events. The evaluation of the solution shows that it is able to extract the causal relations of the vast majority of events occurring in an HPC system. Moreover, its evaluation by administrators validates the highlighted correlations.

Both contributions and their evaluations show the benefit of using machine learning solutions for understanding and predicting failures in HPC systems by automating the analysis of supervision data.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Face à la course à la puissance de calcul, les systèmes informatiques deviennent de plus en plus grands et complexes. Cette course s'illustre particulièrement dans le cas des supercalculateurs (abréviés HPC pour *High Performance Computing* par la suite). En effet dans ces systèmes, la puissance de calcul est multipliée en moyenne par 10 tous les 5 ans [1]. Cette augmentation de la puissance est permise à la fois par l'amélioration des performances matérielles et logicielles, par l'utilisation d'accélérateurs matériels tels que les cartes graphiques mais aussi par l'augmentation du nombre de processeurs.

L'interdépendance de l'ensemble des composants des systèmes HPC tels que le stockage ou le réseau rend le système particulièrement sensible aux défaillances. En effet, à cause de cette interdépendance, la défaillance d'un seul composant peut engendrer une défaillance globale du système. De plus, la fréquence des défaillances est aussi directement liée à la taille et à la complexité de ces systèmes. L'analyse des journaux systèmes sur la machine pétaflopique Blue Waters montre que le temps moyen entre deux défaillances est de l'ordre de 4 heures [24]. Les applications utilisant ces systèmes HPC s'exécutent durant plusieurs dizaines, voire centaines, d'heures. Cette longue durée d'exécution implique une probabilité forte qu'une ou des défaillances se produisent durant l'exécution d'une application. Cependant ces défaillances peuvent amener au dysfonctionnement global de l'application et entrainer son arrêt prématuré. Ces remarques amènent naturellement la question de la gestion aux défaillances d'un tel système.

La gestion des défaillances peut être abordée en suivant plusieurs axes différents de travail. Le premier est de considérer que les défaillances se produisant sur le système font partie intégrante de son fonctionnement, et de mettre au point des stratégies pour mitiger leurs conséquences et leur coût sur le système. Cet axe de travail est actuellement celui adopté par une majorité de systèmes HPC. En effet, des solutions ont été développées pour pouvoir assurer que des applications puissent fonctionner durant des heures sur un nombre de machines conséquent (plusieurs centaines) en fournissant le résultat attendu. Une de ces solutions communément utilisées est la sauvegarde de points de reprise de l'application. L'application sauvegarde régulièrement (toutes les heures par exemple) son état. Si une défaillance survient, l'application peut recharger son dernier point de reprise en ne perdant ainsi que les calculs réalisés depuis la dernière sauvegarde. Cette solution, bien que simple à mettre en place, a néanmoins des désavantages : génération d'un transfert important de données sur le réseau, utilisation importante du stockage, ralentissement de l'application pendant qu'elle crée ses points de reprise, etc. Le deuxième axe de travail est d'analyser les défaillances pour être capable de les comprendre et de les corriger, voire de les prédire.

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

Ces deux axes de travail ne sont pas totalement disjoints. En effet, il est envisageable de tirer parti d'une meilleure compréhension des défaillances pour améliorer la fiabilité des systèmes HPC. Il est par exemple possible d'améliorer les systèmes de sauvegarde actuels de l'application en se focalisant sur la prédiction des défaillances. Si nous sommes capables de prédire les défaillances, une sauvegarde de l'application ne sera déclenchée qu'en cas de défaillance prédite, ce qui permettra de réduire le surcoût de celles-ci [10].

La prédiction et la compréhension de ces défaillances peuvent être réalisées grâce aux données de supervision (*monitoring*) produites par les systèmes HPC. Ces données décrivent l'état du système et contiennent de nombreuses informations concernant les événements se produisant sur le système, l'état des différents composants ou l'exécution des applications par exemple. Ces données peuvent donc décrire directement ou indirectement les défaillances se produisant dans le système.

Cependant, l'analyse des données de supervision est une tâche difficile. En effet, les données de supervision reflètent directement la complexité et la taille des systèmes HPC. Elles sont de différents types (numérique ou textuel par exemple), couvrent différents événements (connexion d'un utilisateur, surchauffe d'un processeur, démarrage d'une application, etc.), proviennent de différentes sources (système de fichiers, système d'exploitation, réseau d'interconnexion, etc.) et ont un volume important (plusieurs téraoctets de données pour un système HPC de taille moyenne par an). De plus, elles sont collectées en suivant les contraintes d'un système HPC. La collecte doit avoir peu d'influence sur la capacité de calcul du système et doit se faire sans modifications majeures des systèmes déjà en place. L'ensemble de ces contraintes induit des données de supervision qui peuvent être peu précises ou avec peu d'informations contextuelles. L'ensemble de ces caractéristiques rend l'analyse difficile par un opérateur humain.

L'apprentissage machine peut nous aider dans cette tâche d'analyse. En effet, il représente l'état de l'art dans de nombreux domaines, tel la classification d'images [98], la détection de personnes [32] ou encore l'analyse de série temporelle [39, 96], grâce à ces capacités de traitement automatisé de grands volumes de données. Nous pouvons l'utiliser pour découvrir automatiquement des corrélations entre les défaillances et les données de supervision. Ces corrélations peuvent ensuite être utilisées de plusieurs manières. Nous pouvons envisager de les utiliser dans le cas de la prédiction des défaillances en utilisant les corrélations pour détecter des événements précurseurs de la défaillance. Nous pouvons également les utiliser dans le cas de la compréhension des défaillances en triant de manière efficace les données de supervision pour ne présenter à l'utilisateur que les informations importantes issues des données corrélées avec la défaillance étudiée.

Exploiter au mieux les techniques d'apprentissage machine pour comprendre et prévoir les défaillances aux travers de l'analyse des données de supervision est une tâche complexe. En effet, il existe de nombreuses techniques d'apprentissage machine avec des spécificités pour chacune de ces techniques. Ces spécifités peuvent porter sur les données utilisées en entrée, la capacité de généralisation de la technique, le volume de données, etc. De plus, les défaillances dans les systèmes HPC sont diverses du fait de la complexité des systèmes qu'ils intègrent et du nombre de composants différents. Chaque type de défaillances nécessite une approche différente qui tient compte de l'ensemble des caractéristiques des données pour obtenir les meilleurs résultats possibles.

Dans cette thèse, nous allons nous focaliser sur l'analyse des données de supervision pour pouvoir comprendre et prédire les défaillances et nous allons montrer qu'il est possible d'utiliser des techniques d'apprentissage machine pour réaliser de manière automatisée cette analyse.

Objectifs et contributions de la thèse

Les travaux présentés dans cette thèse portent sur dans l'automatisation de l'analyse des données de supervision à l'aide de méthodes d'apprentissage machine. Cette analyse a deux objectifs différents : la compréhension des défaillances et la prédiction de celles-ci. Ainsi nous présentons deux contributions distinctes. La première contribution est la mise au point d'une solution automatisée permettant la prédiction d'une défaillance peu étudiée dans l'état de l'art se produisant dans les systèmes HPC : la surchauffe des processeurs. La deuxième contribution est la construction d'une solution automatisée d'aide à l'analyse des journaux systèmes. Les deux contributions sont évaluées notamment à l'aide des données de supervision qui proviennent d'un système HPC utilisé en production par DKRZ [2].

Une solution de prédiction et de correction des surchauffes des processeurs

Cette première contribution se concentre sur une défaillance particulière : la surchauffe des processeurs. La surchauffe des processeurs, bien qu'étant un problème observé dans un supercalculateur, n'est à notre connaissance pas étudiée dans l'état de l'art. En effet, la surchauffe n'amène que dans des cas extrêmes à l'arrêt voire à des dégradations matérielles du processeur. La plupart du temps, le processeur dispose de protections matérielles permettant d'assurer son fonctionnement avec des performances dégradées dans le but de réduire sa température et d'éviter des dommages matériels. Ces protections permettent ainsi à la surchauffe de ne pas avoir de conséquences directes sur le résultat attendu des applications. L'application continue de s'exécuter mais son temps d'exécution augmente.

La prédiction des surchauffes est donc notre objectif principal. Pour être utiles, ces prédictions doivent répondre à certaines contraintes. En effet, pour envisager de construire une solution capable de réduire efficacement l'influence des surchauffes sur le système, il est nécessaire d'être capable de prédire les surchauffes avec quelques minutes d'avance pour pouvoir prendre des actions préventives, tout en limitant les faux positifs. De plus, le surcoût du fonctionnement de la solution pour le système devra être le plus faible possible.

Comme expliqué précédemment, pour être capable de prédire efficacement une défaillance, il est nécessaire de concevoir une approche adaptée aux données de supervision utilisées et aux objectifs fixés. Ainsi, nous présentons une analyse des conditions d'apparition des surchauffes dans un système HPC. Cette analyse met en évidence plusieurs points clés à prendre en compte pour la conception de la solution prédictive. Le premier point est que les conditions d'apparition des surchauffes des processeurs dans un système HPC font intervenir un nombre important de paramètres et que la seule donnée disponible dans les données de supervision analysées capable de décrire efficacement les surchauffes est la température des processeurs. Le deuxième point est que les surchauffes influent de manière importante sur le temps d'exécution des applications HPC car elles sont responsables d'une baisse de fréquence du processeur.

Ces deux points nous permettent ensuite de construire une solution de prédiction des surchauffes et d'évaluer sa pertinence. La solution de prédiction proposée repose sur l'analyse de séries temporelles représentant la température des processeurs. L'analyse des séries temporelles est réalisée en tenant compte de la forme globale des séries temporelles avant un événement de surchauffe. Grâce à cette prise en compte de la forme, nous sommes capables de détecter les schémas du comportement de la température avant un événement

2. Centre allemand de recherche sur la climatologie https ://www.dkrz.de/

de surchauffe sur de grandes périodes de temps, et ainsi d'obtenir des prédictions fiables plusieurs minutes en avance.

De plus, nos évaluations mettent en évidence la supériorité de notre solution par rapport à des approches essayant de prédire la température des processeurs dans les prochaines minutes.

La fiabilité des prédictions fournies par la solution est évaluée en estimant la réduction de l'influence des surchauffes grâce à la prise d'actions préventives, telles qu'une légère baisse de fréquence anticipée, faites en fonction des prédictions réalisées. Les résultats montrent une réduction significative de l'influence des surchauffes dans un système HPC, ce qui nous permet de valider notre approche et notre solution.

Une solution d'aide à l'analyse des journaux systèmes

Cette deuxième contribution s'intéresse à la mise au point d'une solution d'aide à l'analyse des journaux systèmes. L'analyse de ces journaux est importante car ils contiennent de précieuses informations sur l'état du système. En effet, tous les services qui s'exécutent sur un composant d'un système HPC écrivent des informations concernant leurs états dans un même journal système. L'analyse peut donc nous apporter des informations sur l'ensemble des étapes menant à une défaillance. Cependant, cette analyse est difficile pour un opérateur humain car l'utilisation du même journal système induit un entrelacement permanent des informations provenant de l'ensemble des services. Cet entrelacement complique la découverte des corrélations entre les différentes actions prises par les services car un nombre important d'informations non corrélées peut être écrit entre deux informations corrélées. De plus, les informations écrites peuvent être très variées (connexion d'un utilisateur, surchauffe d'un processeur, démarrage d'une application, etc.).

Pour aider à l'analyse de ces journaux et à la mise en évidence des corrélations possibles entre les différentes informations présentes dans les journaux systèmes, nous proposons une solution nommée LogFlow. Cette solution fournit à l'utilisateur une vue augmentée de ces informations. Cette vue permet non seulement de n'afficher que les informations pertinentes à propos de la défaillance ou de l'événement sélectionné mais aussi de montrer les liens causaux entre les différentes informations. Elle repose sur une approche découpée deux grandes étapes : la découverte des événements dans les journaux systèmes permettant l'utilisation de techniques d'apprentissage machine et l'apprentissage en lui-même des liens entre les événements.

La découverte des événements dans les journaux systèmes est un domaine très étudié dans l'état de l'art et il existe de nombreuses solutions pour répondre à cette problématique. Cependant, l'étude des solutions proposées montre qu'elles ne sont pas capables de traiter des grands volumes de données, tels que ceux produits par les systèmes HPC, en un temps *raisonnable*. La solution que nous proposons permet une analyse des journaux de très grandes tailles provenant des systèmes à large échelle en *temps réel*. En effet, grâce à des suppositions fortes mais réalistes sur la structure des journaux systèmes des systèmes HPC, nous proposons une solution qui permet un traitement entièrement parallèle des entrées contenues dans les journaux. De plus, l'utilisation de représentations efficaces de ces entrées nous permet de réduire l'empreinte mémoire de la solution. L'ensemble de ces caractéristiques permet à notre solution de pouvoir traiter plusieurs millions d'entrées par secondes ce qui représente une amélioration de plusieurs ordres de grandeur par rapport aux meilleures solutions de l'état de l'art.

L'apprentissage des liens de causalité entre les événements présents dans les journaux

systèmes utilise un modèle d'apprentissage profond. L'utilisation d'un modèle d'apprentissage profond dans cette tâche de découverte des corrélations n'est pas conventionnelle. En effet, ce type de modèle est classiquement utilisé pour des tâches de prédiction mais non pour des tâches de mise en évidence des corrélations. Pour parvenir à découvrir les corrélations à l'aide de ce modèle d'apprentissage profond, une nouvelle couche est ajoutée à ce modèle. Cette couche s'appuie sur des travaux récents de l'état de l'art qui proposent une nouvelle couche, appelée couche d'attention, permettant de connaitre l'importance des entrées utilisées par le modèle pour réaliser sa prédiction. Grâce à cette information, il est possible de déduire les corrélations entre ces entrées et la prédiction.

L'évaluation de LogFlow montre qu'il est capable de fournir des résultats fiables et de mettre en évidence des corrélations entre une grande partie des événements survenant dans un système HPC. De plus, le temps d'exécution de LogFlow le rend compatible avec une utilisation interactive par un utilisateur et son automatisation totale rend son intégration et son utilisation facile.

Organisation du document

Ce document s'organise de la façon suivante. Le chapitre 2 introduit les notions essentielles autour des systèmes HPC et de la gestion des défaillances. Nous expliquons ensuite l'apport possible de l'apprentissage machine pour la prédiction ou la compréhension des défaillances. Le chapitre 3 étudie les surchauffes des processeurs. Il présente une analyse des causes et des conséquences d'un tel événement dans un système HPC. Le chapitre 4 décrit notre solution permettant de prédire les événements de surchauffe. Il présente également une évaluation des capacités prédictives de notre solution et une évaluation pratique en incluant des actions préventives. Le chapitre 5 décrit la solution d'aide à l'analyse des journaux systèmes et les résultats pratiques de la solution. Nous concluons par le chapitre 6 qui présente un bilan de nos travaux et les différentes perspectives envisagées.

Chapitre 2

Défaillances dans les systèmes haute performance

A travers ce chapitre, nous abordons la problématique des défaillances dans les systèmes HPC.

De nombreux domaines nécessitent d'importantes capacités de calcul. Citons par exemple la prévision météorologique, la simulation nucléaire ou encore l'intelligence artificielle. Pour répondre à ces besoins, des systèmes spécialisés pour ce type de calcul sont utilisés : les systèmes HPC. Ces systèmes sont entièrement pensés pour offrir le maximum de performance à l'utilisateur et ces hautes performances se doivent d'être garanties. En effet, le coût financier d'un système HPC est important. Actuellement, le plus puissant de ces systèmes est Summit [1]. Il est hébergé par le laboratoire d'Oak Ridge aux Etats-Unis. Son coût est estimé à plus de 300 millions d'euros. Sa puissance de calcul est équivalente à plus de 32 000 stations de travail.

Au vu du coût financier de ces systèmes et la fréquence importante des défaillances [24], il est nécessaire d'utiliser des solutions permettant de gérer les défaillances pour réduire leurs conséquences sur le système. Pour décrire plus précisément ce besoin, nous décrivons dans un premier temps les caractéristiques de ces systèmes afin de comprendre leurs spécificités. Nous détaillons ensuite la gestion des défaillances dans ces systèmes et nous décrivons brièvement les solutions actuellement utilisées. Puis nous abordons les travaux autour de l'amélioration de ces solutions qui utilisent les techniques d'apprentissage machine pour l'automatisation de l'analyse des données de supervision. Nous décrivons ensuite le fonctionnement de ces techniques d'apprentissage machine et comment elles peuvent nous aider dans cette automatisation. Enfin, nous concluons en expliquant les défis liés à l'utilisation des techniques d'apprentissage machine pour l'automatisation de l'analyse des données de supervision.

2.1 Systèmes HPC

Pour mieux comprendre la problématique de la gestion des défaillances dans un système HPC, il est nécessaire d'en connaître les caractéristiques. Dans cette section, nous décrivons brièvement l'architecture générique d'un système HPC, puis nous expliquons chaque grande brique technologique de cette architecture : les nœuds de calcul, le système de fichiers, le réseau d'interconnexion, les logiciels et enfin les applications scientifiques.

1. https ://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

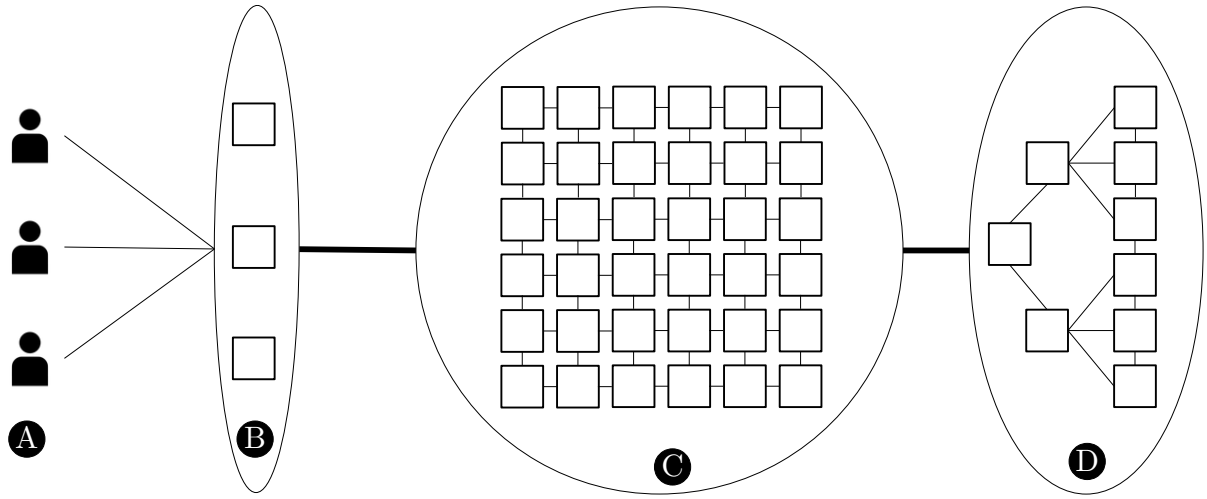


FIGURE 2.1 – Structure simplifiée d'un système HPC.

Nous décrivons ensuite la collecte des données de supervision utilisée pour connaître l'état de fonctionnement du système. Ces descriptions nous permettront de comprendre, dans un second temps, les problématiques liées à la gestion des défaillances dans un tel système.

2.1.1 Architecture générique d'un système HPC

Un système HPC est composé de plusieurs ordinateurs distincts (communément appelés nœuds). Chaque nœud est relié aux autres par un réseau d'interconnexion. Pour atteindre les meilleures performances possibles, chaque nœud est dédié à une tâche précise (calcul, système de fichiers, connexion des utilisateurs, affichage des résultats, etc.) car chaque type de tâche requiert une configuration particulière pour être exécutée le plus rapidement possible. Un système HPC est donc composé de plusieurs types de nœuds, chacun ayant une configuration matérielle et logicielle spécifique. Ainsi, les nœuds dédiés au calcul sont les nœuds offrant la plus grande puissance de calcul, les nœuds dédiés au système de fichier le plus grand stockage, etc.

Cependant un système HPC n'est pas composé uniquement de nœuds spécifiques, c'est aussi un ensemble de logiciels dédiés à son fonctionnement. La figure 2.1 présente la structure simplifiée d'un système HPC. Cette représentation est volontairement simplifiée et ne fait apparaître qu'une partie de l'architecture réelle d'un système HPC. La partie A représente les utilisateurs du système, la partie B représente les nœuds qui sont en charge de la réservation et de l'exploitation des ressources, la partie C représente les nœuds de calcul et la partie D représente les nœuds dédiés au système de stockage. Cette structure fait apparaître le mode de fonctionnement commun d'un système HPC : les utilisateurs (A) via des nœuds en charge de la réservation des ressources (B) réservent une partie des ressources disponible (C et D) pour exécuter une application. Cette application utilise ensuite les ressources réservées (C) pour s'exécuter.

2.1.2 Nœuds de calcul

Le cœur d'un système HPC réside dans ses nœuds de calcul. Ces nœuds sont dédiés aux différents types de calcul requis par les applications pour, par exemple, réaliser une prévision météorologique. Ces nœuds sont caractérisés par la présence d'un ou deux processeurs disposant d'un nombre élèves de cœurs et d'une quantité importante de

Type de composant	Nœud de calcul de Summit	Station de travail
Nombre de processeurs	2	1
Cœurs de calcul	44	8
Mémoire Vive	512 Go	8Go
Accélérateurs	6 *GPU*	1 *GPU*
Puissance de calcul (TFlops)	42	5

Tableau 2.1 – Comparaison entre un nœud de calcul du supercalculateur Summit et une station de travail.

mémoire vive. Selon le type de configuration, nous pouvons aussi retrouver des accélérateurs matériels particuliers pour certains types de calcul spécifiques comme par exemple les cartes graphiques (*GPU*) ou les circuits programmables (*FPGA*). Un nœud de calcul est la plupart du temps composé de deux processeurs haute performance avec quelques dizaines de cœurs et une centaine de gigaoctet de mémoire vive. Le tableau 2.1 fournit une comparaison entre une station de travail et un nœud de calcul de Summit qui est le plus puissant système HPC construit à ce jour. Nous pouvons très clairement voir qu'un seul nœud de calcul de Summit est environ 8 fois plus puissant qu'un ordinateur classique et qu'il dispose en plus d'accélérateurs spécifiques (6 *GPU*).

2.1.3 Système de fichiers

Les systèmes de fichiers dans les systèmes HPC permettent de stocker de très grands volumes de données (plusieurs pétaoctets) avec des débits d'accès à ces données très élevés (plusieurs centaines de gigaoctets par seconde). Il est donc nécessaire pour garantir ces performances de faire appel à un système de fichiers distribué physiquement distinct des nœuds de calcul. Cela implique qu'il n'y a pas un seul nœud dédié au stockage mais des dizaines de nœuds travaillant en parallèle. Cette distribution permet à la fois de gagner en performance en pouvant utiliser plusieurs nœuds en parallèle pour accéder à plusieurs fichiers mais également en fiabilité en permettant la redondance (un fichier est présent sur plusieurs nœuds simultanément). Cette redondance permet d'éviter de perdre les données présentes sur un disque dur dans le cas de la défaillance de celui-ci car elles sont répliquées sur d'autres disques durs dans le système.

2.1.4 Réseau haute performance

Pour relier l'ensemble des composants du système HPC, il est nécessaire de faire appel à un réseau haute performance. Celui-ci se charge de la communication entre les nœuds de calcul afin qu'ils puissent échanger des données dans le cas où une application utilise plusieurs nœuds de calcul en parallèle mais également de faire communiquer les nœuds de calcul avec le système de fichiers.

Il est composé de fibre optique pour garantir des débits élevés et une latence minimale. Enfin, l'architecture du réseau est également pensée pour la performance en permettant par exemple d'optimiser la latence en réduisant au maximum la longueur du câble entre deux nœuds physiquement proches.

Plusieurs types d'architectures sont utilisés. La figure 2.2 présente un type d'architecture utilisée : le fat-tree. Dans cet exemple, le réseau est représenté sous la forme d'un arbre

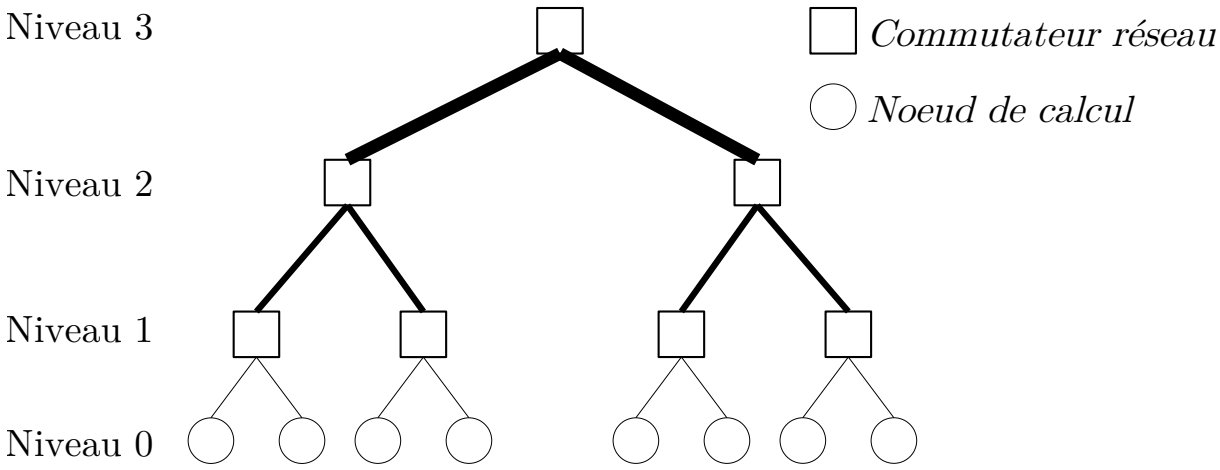


FIGURE 2.2 – Exemple d'une architecture réseau d'un système HPC : le fat-tree.

binaire : un commutateur réseau est connecté à deux autres équipements descendants (nœud de calcul ou autre commutateur). Les feuilles (niveau 0) représentant les nœuds de calcul sont reliées par un lien simple. A chaque nouveau niveau atteint, les connexions sont doublées. Entre le niveau 1 et le niveau 2, chaque commutateur dispose d'une double connexion, entre le niveau 2 et 3 chaque commutateur dispose de 4 connexions et ainsi de suite. L'objectif principal de ce type d'architecture est d'augmenter la bande passante globale et de permettre l'utilisation de plusieurs liens simultanément. Cela permet aussi d'avoir des connexions proches rapides car elles ne nécessitent que de traverser un commutateur réseau dans le meilleur des cas. À l'inverse, les communications lointaines (*i.e.* devant utilisées plusieurs commutateurs réseaux) sont plus longues car il est nécessaire de travers de nombreux commutateurs réseaux.

De nombreuses solutions techniques sont utilisées pour réaliser le réseau d'interconnexion d'un système HPC, nous pouvons citer *Infiniband*[2] ou encore *Omnipath*[3]

2.1.5 Logiciels

Des logiciels spécifiques sont nécessaires pour exploiter l'ensemble d'un système HPC. Nous n'allons pas dresser une liste exhaustive des logiciels utilisés, mais simplement citer les plus importants.

Pour commencer, un gestionnaire de ressources est nécessaire. En effet, lorsqu'un utilisateur souhaite utiliser un ensemble de ressources, celles-ci doivent être configurées préalablement en fonction de sa demande. Cependant, il est complexe de demander à un utilisateur de configurer lui-même des ressources comme les nœuds de calcul ou les accélérateurs spécifiques avant de pouvoir les utiliser. De plus, l'ensemble des ressources sont distribuées entre plusieurs utilisateurs. Il est donc nécessaire d'avoir un gestionnaire de ressources qui va attribuer à chaque utilisateur les ressources selon sa demande et la disponibilité des ressources dans le système. Le gestionnaire va ensuite s'assurer que les ressources demandées soient configurées conformément à la demande de l'utilisateur. *Slurm*[4] peut être cité comme exemple de gestionnaire de ressources.

2. https ://www.infinibandta.org/
3. https ://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html
4. https ://slurm.schedmd.com/

Un autre aspect nécessitant un logiciel spécifique est le système de fichiers. Comme nous l'avons expliqué précédemment, le système de fichiers est composé de plusieurs dizaines de nœud de stockage et les données sont réparties sur l'ensemble des nœuds de stockage. Un logiciel s'occupe de fournir un vue classique du système de fichiers à partir des dizaines de nœuds de stockage dont il dispose. Ce logiciel se charge de répartir les fichiers sur les nœuds de stockage pour permettre les meilleures performances possibles mais aussi pour garantir la redondance des fichiers. Nous citerons *Lustre*[5] comme exemple de système de fichiers.

Enfin, pour que les applications puissent utiliser plusieurs nœuds de calcul en parallèle, des surcouches logicielles sont utilisées. En effet, l'utilisation de plusieurs nœuds de calcul en parallèle qui communiquent à travers le réseau d'interconnexion n'est pas facile. En effet, pour qu'une application puisse effectuer des calculs en parallèle sur plusieurs nœuds distincts, il est nécessaire la plupart du temps d'échanger des données entre ces nœuds. Si nous découpons un problème en 4 parties, il est rare que les 4 parties soient totalement indépendantes. Par exemple, le calcul des prévisions météorologiques peut être *facilement* parallélisable. Si nous devons réaliser la prévision météorologique pour toute la France, nous pouvons la découper selon les régions[6]. Les prévisions de chaque région sont calculées sur un nœud de calcul distinct en parallèle et les prévisions sont ensuite agrégées au niveau national. Nous avons ainsi autant de nœuds de calcul utilisés que de régions. Cependant, le calcul de la prévision d'une région fait appel aux prévisions des régions frontalières : une perturbation météorologique se déplaçant de région en région par exemple et qui influe sur les prévisions des régions qu'elle traverse. Les différentes régions doivent donc échanger des données pendant le calcul des prévisions. Cet échange de données doit être réalisé de la manière la plus rapide possible sous peine de ralentir le calcul en attendant sans cesse des données venant des régions frontalières. Une surcouche logicielle pour les applications est donc fournie pour permettre de prendre en charge cet échange de données en le faisant de la manière la plus rapide possible en utilisant le réseau d'interconnexion haute performance. Il est à noter que les résultats de cette surcouche logicielle dépendent aussi de son intégration dans l'application par les développeurs. Un exemple commun de cette surcouche est *Open MPI*[7].

2.1.6 Applications et utilisation d'un système HPC

La dernière brique d'un système HPC est les applications. En effet, non seulement le système entier est tourné vers la haute performance mais les applications doivent également être conçues dans ce sens. Au-delà de l'optimisation *classique* des applications et de l'utilisation de langage de bas niveau (majoritairement *C-C++* et *Fortran*), l'application doit également utiliser le système de manière efficace.

Pour illustrer le besoin d'avoir une utilisation efficace, reprenons l'exemple d'une application de calcul de prévisions météorologiques. Comme expliqué précédemment, le calcul peut être distribué sur plusieurs nœuds de calcul distincts, chacun s'occupant d'une région de la France. Les nœuds de calcul doivent communiquer entre eux pour échanger les prévisions météorologiques aux frontières des régions et ainsi fournir une prévision globale cohérente et non une prévision locale indépendante des autres régions.

Cette communication doit se faire de manière performante pour garantir un temps

5. http ://lustre.org/
6. Un tel découpage n'est pas réaliste en pratique mais nous l'utilisons pour illustrer le problème.
7. https ://www.open-mpi.org/

d'exécution de l'application minimal. Considérons que les calculs sont équitablement répartis entre les nœuds. Si un nœud de calcul est plus lent à communiquer avec les autres à cause d'un problème sur le réseau par exemple, il ralentit ses voisins proches (les régions frontalières dont la prédiction dépend de celle-ci) mais également l'ensemble des nœuds par rebond. En effet, si les nœuds proches sont ralentis en attendant le résultat de ce nœud, ceux-ci ralentissent à leur tour leurs nœuds voisins créant ainsi un effet boule de neige : l'ensemble des nœuds se retrouve à attendre le nœud avec les communications les plus lentes.

Ce simple exemple nous permet de comprendre pourquoi l'application doit également utiliser le système HPC de manière efficace : l'efficacité des communications entre les nœuds doit intégrer à la fois lors du développement de l'application, lors de la conception du système HPC mais aussi lors de l'utilisation de l'application sur le système HPC. En effet, il sera nécessaire de regrouper au maximum les nœuds utilisés par l'application pour bénéficier de latence minimale entre les nœuds. Il est à noter que ce regroupement des nœuds utilisés par l'application est à la charge du gestionnaire de ressources.

2.1.7 Collecte des données de supervision

Pour permettre de superviser l'état d'un système HPC, des données de supervision sont collectées sur l'ensemble des composants du système. À des intervalles de temps régulier chaque composant du système HPC est surveillé et des métriques sont remontées pour indiquer l'état de celui-ci. Nous pouvons donner comme exemple la collecte de la bande passante du réseau, ou de l'espace de stockage disponible sur le système de fichiers. Du fait de la complexité du système, des milliers de métriques sont collectées à des intervalles de temps variables (de la seconde à la minute). En complément de ces métriques, les différents composants peuvent écrire des entrées dans les journaux systèmes pour indiquer des événements (l'écriture d'un fichier ou le démarrage d'une application par exemple). Ces deux sources d'information sont les principales sources pour permettre de connaître l'état de santé d'un système HPC et pour pouvoir analyser et diagnostiquer les défaillances.

Il est à noter que ces deux sources sont corrélées : le réseau écrira une entrée dans les journaux systèmes si la bande passante disponible est faible et la métrique collectant la bande passante disponible nous fournira la même information. La métrique nous permettra d'être le plus précis en suivant l'évolution de la banque passante alors que l'entrée dans les journaux systèmes nous permettra d'être informée plus facilement en lisant l'entrée et l'événement correspondant.

2.1.8 Synthèse

Nous avons vu à travers cette description que les conditions nécessaires à la garantie d'une haute performance des systèmes HPC sont nombreuses et variées. Ainsi, chaque goulet d'étranglement doit être analysé et réduit au minimum. À l'ensemble de ces conditions vient s'ajouter la taille des systèmes. En effet, il peut paraître simple de gérer une centaine de nœuds, mais les systèmes HPC actuels sont composés de plusieurs millions de cœurs, de plusieurs milliers de nœuds et utilise l'équivalent en électricité d'une ville de 20 000 habitants [8]. La problématique de l'exploitation des ressources de manière efficace est donc centrale.

8. https ://www.top500.org/system/179397

Cependant, cette exploitation efficace des ressources peut être dégradée par des défaillances survenant dans le système. En effet, si une défaillance survient lorsqu'une application est en cours de fonctionnement, il est probable que cela empêche d'obtenir le résultat final. Il sera donc nécessaire de redémarrer l'application depuis le début. Dans ce cadre, et pour améliorer la fiabilité des systèmes, des méthodes peuvent être mises en place pour tenter de réduire le coût des défaillances dans les systèmes HPC. Dans la prochaine partie, nous présentons différentes approches utilisées pour la gestion des défaillances dans les systèmes HPC.

2.2 Gestion des défaillances dans les systèmes HPC

Nous avons vu que les systèmes HPC sont un ensemble complexe de nombreux composants différents. Ces composants sont connectés et sont utilisés simultanément afin de fournir la meilleure performance. Cette interconnexion ajoutée à la taille des systèmes rend les défaillances fréquentes.

De nombreuses études récentes montrent que les défaillances peuvent être fréquentes dans un système HPC. Di Martino et al. [24] montrent que le temps moyen entre deux défaillances est de 4 heures sur la machine pétaflopique Blue Waters. Philp et al [73] estiment le temps moyen entre deux défaillances à 1 heure et 30 minutes sur un système pétaflopique. Wang et al. [92] donnent un intervalle de variation du temps moyen entre deux défaillances compris entre 6,5 et 40 heures selon l'âge et la maturité du système étudié.

De plus, la croissance rapide des systèmes HPC en taille pour atteindre l'exaflop (un milliard de milliard d'opérations par secondes) complique et aggrave les conséquences des défaillances. En effet, Di Martino et al. [25] montrent, à travers l'analyse de plus de 5 000 000 millions de démarrages d'applications, que la probabilité de subir les conséquences d'une défaillance est multipliée par 20 pour une application sur 20 000 nœuds par rapport à une application déployée sur 10 000 nœuds. Ceci s'explique par l'interconnexion de l'ensemble des composants : la probabilité de défaillance d'un nœud sur l'ensemble des nœuds augmente rapidement avec l'augmentation du nombre de nœuds utilisés.

Ces études mettent en exergue le besoin d'avoir des solutions permettant la gestion des défaillances car elles se produisent donc de manière quasi-quotidienne sur les systèmes HPC.

La gestion des défaillances peut se reposer sur deux grandes approches : supposer que les défaillances se produisent et trouver des méthodes pour être tolérant à ces défaillances, ou prédire et corriger les défaillances avant qu'elles ne se produisent. Ces deux méthodes ne sont pas disjointes, la prédiction des défaillances peut permettre d'améliorer et de simplifier les systèmes de tolérance aux défaillances.

Nous commençons cette section par donner les caractéristiques des défaillances survenant dans un système HPC et leur influence sur la fiabilité d'un système HPC. Puis, nous abordons les techniques actuelles de tolérance aux défaillances. Nous présentons ensuite les travaux réalisés dans l'état de l'art pour améliorer la gestion et la tolérance aux défaillances en analysant les données de supervisions pour comprendre et prédire les défaillances.

2.2.1 Caractéristiques et conséquences d'une défaillance

Commençons tout d'abord par présenter les caractéristiques d'une défaillance dans un système HPC et détailler ses conséquences sur le système. Les défaillances peuvent se

produire sur plusieurs composants d'un système HPC : les nœuds de calcul, le système de fichiers, le réseau haute performance, etc. Chacun d'entre eux peut subir des défaillances de différents types : une défaillance d'un disque dur peut se produire sur le système de fichiers (le disque devient alors inopérant), une surchauffe peut se produire sur le processeur d'un nœud de calcul, un câble réseau peut être débranché ce qui réduira les performances du réseau, etc.

Le classement et l'analyse des défaillances dans les systèmes HPC est un domaine largement traité dans l'état de l'art [15, 23, 40, 79]. Nous proposons ici un court résumé. Pour cela, nous n'allons pas lister l'ensemble des défaillances mais les caractériser brièvement pour mieux comprendre leurs conséquences possibles. Nous choisissons ici de les présenter et de les caractériser selon deux grandes classes : les défaillances *visibles* d'un point de vue utilisateur, et les défaillances *invisibles* pour l'utilisateur. Il est à noter qu'il existe des classements utilisant d'autres caractéristiques des défaillances. Par exemple Gupta et al. [40] proposent de se focaliser sur l'origine des défaillances (matérielle ou logicielle) et le composant mis en cause (processeur, carte graphique, système de fichiers, etc.).

Les défaillances *visibles* d'un point de vue des utilisateurs sont par exemple le dysfonctionnement d'une alimentation électrique entraînant l'arrêt d'un ou plusieurs nœuds de calcul ou la défaillance d'une barrette de mémoire vive. La principale conséquence est que l'application utilisant le nœud subissant la défaillance sera arrêtée et que l'utilisateur ne pourra pas obtenir le résultat de l'application si aucune action n'est prévue pour remédier à la défaillance.

Les défaillances *invisibles* pour l'utilisateur peuvent être la surchauffe d'un processeur par exemple. Le processeur subissant une surchauffe voit ses capacités de calcul être réduites pour réduire sa température. Cette réduction entraîne un temps d'exécution plus long de l'application que dans des conditions normales. Néanmoins, l'application continue de fonctionner. Un autre exemple peut être celui de l'inversement de la valeur d'un bit. Cette défaillance entraîne une erreur dans les calculs réalisés par l'application. Cette erreur peut soit se propager de manière à invalider le résultat, soit être minimisée par les calculs suivants de l'application et ne pas influencer le résultat final. Cette inversion de bit n'est pas détectable directement. Il est nécessaire de comparer deux résultats du même calcul pour la découvrir.

Nous voyons donc que les défaillances peuvent avoir des caractéristiques différentes et que leur coût pour le système n'est pas le même. De plus, leur détection n'est pas tout le temps immédiate car elles peuvent être invisibles pour l'utilisateur du système. Cependant, plusieurs approches pour rendre les applications tolérantes aux défaillances peuvent être mises en place. Nous allons maintenant détailler quelques exemples de méthodes pouvant être utilisées pour rendre les applications tolérantes aux défaillances.

2.2.2 Tolérance aux défaillances

La tolérance aux défaillances dans les systèmes HPC est un sujet largement traité par l'état de l'art [16, 46, 31, 15]. La tolérance aux défaillances repose sur le fait de supposer qu'une défaillance se produira probablement durant l'exécution de l'application et qu'il est nécessaire trouver une solution pour que l'application puisse continuer à s'exécuter malgré la défaillance.

Une des solutions les plus simples est de démarrer plusieurs instances de l'application en même temps et de comparer les résultats finaux. Ainsi, si une défaillance est survenue sur un des composants du système HPC, il suffit de comparer et de garder les résultats

d'une des instances de l'application. Le principal défaut de ce type de solution est son coût. Le démarrage de multiples instances de l'application mobilise un nombre important de ressources uniquement dans le but d'assurer l'obtention d'un résultat. Et ces ressources ne peuvent pas être utilisées par un autre utilisateur. Ferreira et al. [33] proposent une évaluation du coût de la redondance pour la gestion des défaillances. Ils concluent que ce type de méthode est viable pour des systèmes exaflopiques mais qu'il peut cependant exister des méthodes plus efficaces.

D'autres approches sont proposées dans l'état de l'art pour essayer de réduire les ressources nécessaires. Une des solutions les plus communément employées est la sauvegarde de l'état de l'application (*checkpointing*) [43]. L'application sauvegarde à des intervalles de temps spécifique son état. Si une défaillance survient, il suffit de redémarrer l'application en utilisant la dernière sauvegarde. Bien que cette solution permette d'utiliser moins de ressources que le démarrage en parallèle de plusieurs instances de l'application, elle a de nombreux désavantages : le coût de stockage des points de sauvegarde, la consommation de bande passante du réseau lors de la sauvegarde, le ralentissement de l'application lors de la sauvegarde, le temps perdu depuis la dernière sauvegarde, etc. Ainsi Elliot et al.[30] montrent que l'utilisation d'une solution utilisant uniquement la sauvegarde de l'ensemble des points de reprise de l'application sur le système de fichiers centralisé comme solution de tolérance aux défaillances n'est pas viable sur des systèmes à très large échelle. En effet, alors que le coût de celle-ci sur les ressources n'est que de 4% pour un système utilisant 100 nœuds, celui-ci augmente à plus de 65% des ressources pour un système utilisant 100 000 nœuds.

Des solutions ont donc été développées pour réduire le coût de la sauvegarde des applications. Ces solutions peuvent utiliser plusieurs approches différentes. Une des approches peut être d'améliorer le principe de sauvegarde de l'application en assurant le confinement des retours arrières en utilisant des protocoles hiérarchiques combinant sauvegarde de points de reprise et enregistrement de messages [74, 11, 75] ou en utilisant des techniques avancées de sauvegarde des points de reprise en utilisant des techniques d'encodage et la mémoire des nœuds voisins [9, 66].

Une des autres approches que nous pouvons envisager est d'analyser les défaillances. Cette analyse peut être utilisée de plusieurs façons. La première utilisation de cette analyse peut être pour mieux comprendre les défaillances afin mitiger leurs conséquences sur le système. Cependant, les résultats de cette analyse peuvent aussi servir à prédire les défaillances afin de les corriger avant qu'elles ne surviennent à l'aide d'actions préventives ou pour permettre aux solutions de tolérance aux défaillances d'utiliser les prédictions réalisées pour être plus efficace. Nous allons détailler les moyens utilisés pour réaliser l'analyse des défaillances dans un système HPC dans la prochaine partie.

2.2.3 Analyse et prédiction des défaillances en utilisant les données de supervision

Une approche complémentaire de la gestion des défaillances peut être d'analyser et de prédire les défaillances survenant dans les systèmes HPC. Cette approche utilise principalement les informations contenues dans les données de supervision pour réaliser l'analyse et la prédiction des défaillances. Nous allons montrer dans un premier temps, à l'aide d'analyses réalisées sur les données de supervision par des travaux de l'état de l'art, que ces données contiennent des informations pouvant être utilisées pour décrire et mieux comprendre les défaillances. Nous montrons dans un second temps que ces informations

peuvent aussi être utilisées pour prédire les défaillances en détaillant des travaux de l'état de l'art qui proposent des solutions permettant de prédire certains types de défaillances en utilisant les données de supervision. Enfin, en utilisant les résultats de travaux de l'état de l'art, nous présentons les apports possibles de la prédiction des défaillances dans l'amélioration des solutions actuelles de gestion des défaillances.

Analyse des données de supervision pour la compréhension des défaillances

Les données de supervision contiennent de nombreuses informations qui peuvent être utilisées pour mieux comprendre, caractériser ou identifier l'influence des défaillances sur le système. En effet, de nombreux travaux s'appuient sur l'analyse des données de supervision pour caractériser les défaillances.

Gupta et al. [40] présentent une analyse des défaillances survenant dans le système HPC d'Oak Ridge National Laboratory en utilisant les informations présentes dans les journaux systèmes. Grâce à ces informations, ils réalisent plusieurs analyses pour caractériser les défaillances. Par exemple, ils mettent en évidence que les défaillances qui ont pour origine des erreurs matérielles sont prédominantes dans leur système HPC. Di Martino et al. [25] analysent les données de supervision du système HPC Blue Waters pour évaluer la probabilité qu'une défaillance survienne lors de l'exécution d'une application en fonction du nombre de nœuds utilisés par celle-ci. De la même façon, Schroeder et al. [79] utilisent les données de supervision pour caractériser les défaillances et présentent, par exemple, la distribution des défaillances en fonction de l'âge du système HPC. Di et al. [23] utilisent les données de supervision provenant du système HPC Blue Gene/Q pour mettre en évidence que la probabilité d'apparition d'une défaillance dans le système suit une loi de probabilité de type Pareto.

L'ensemble de ces travaux nous permet de conclure que les données de supervision contiennent des informations permettant de décrire et de mieux comprendre les défaillances.

Prédictions des défaillances

En complément des travaux autour de l'analyse des données de supervision, de nombreux travaux se focalisent sur la prédiction des défaillances à l'aide des données de supervision. Cette prédiction repose, la plupart du temps, sur l'utilisation de méthode d'apprentissage machine pour automatiser la recherche de corrélations entre les données de supervision et les défaillances.

Tuncer et al. [85] et Netti et al. [69] proposent d'utiliser différentes métriques collectées sur le système HPC Volta comme le taux d'utilisation du processeur, le nombre de paquets reçus sur le réseau ou la consommation énergétique en entrée d'une méthode d'apprentissage machine pour prédire les défaillances. Ils évaluent la pertinence de plusieurs types de méthodes d'apprentissage tels que les arbres de décision, l'algorithme Adaboost, ou encore les forêts aléatoires. Leurs résultats sur des données synthétiques montrent que leur approche parvient à détecter environ 98% des défaillances. Ils montrent aussi que les différentes méthodes d'apprentissage machine évaluées ont chacune des avantages selon les caractéristiques souhaitées de la solution (taux de faux positifs, rappel, type d'applications considérées, etc.).

Nie et al. [70] s'intéressent aux différents facteurs tels que la température, la consommation électrique ou encore les caractéristiques de la charge de travail pouvant augmenter la probabilité d'apparition de défaillances sur les cartes graphiques. En utilisant ces facteurs, ils construisent une solution utilisant une méthode d'apprentissage machine capable

de détecter 87% des défaillances avec une précision de 76%. De la même façon que les travaux précédents, l'évaluation de plusieurs méthodes d'apprentissage machine telles que les Gradient Boosting Tree, Support Vecteur Machine (SVM) ou encore les réseaux de neurones, montre que les différentes méthodes présentent chacune des avantages différents selon les caractéristiques recherchées (précision, rappel, temps d'entrainement, etc.).

Das et al. [21] proposent une approche permettant de prédire les prochaines défaillances décrites dans les journaux systèmes en utilisant une méthode d'apprentissage machine profond (*Deep learning*). En utilisant les journaux systèmes provenant d'un système HPC en production, ils montrent que leur approche permet de prédire 83% des défaillances avec 2 minutes d'avance tout en gardant une précision de 96%

De la même manière Zheng et al. [106] utilisent les journaux systèmes provenant du système HPC Blue Gene/P pour construire une approche utilisant une méthode d'apprentissage machine pour prédire les défaillances survenant dans le système. Les défaillances sont identifiées dans les journaux systèmes avec l'aide des administrateurs du système HPC. Ils évaluent de manière plus précise une méthode d'apprentissage machine utilisant les algorithmes génétiques. Leurs résultats montrent qu'ils sont capables de prédire 70% des défaillances avec 5 minutes d'avance et avec une précision de 40%

Enfin, Gainaru et al. [37] proposent une méthode pour analyser les corrélations entre les différents événements présents dans les journaux systèmes. Cette méthode repose sur la détection d'événements survenant fréquemment en même temps en s'appuyant sur une version modifiée de l'algorithme Apriori [2]. La méthode proposée présente l'avantage d'utiliser une fenêtre dynamique de temps pour la recherche de corrélations entre les événements mais aussi une mise à jour continue de la méthode dans le temps pour prendre en compte l'apparition de nouvelles corrélations.

De manière plus générique, Aué. [4] propose une étude fournissant une vue d'ensemble des méthodes permettant d'analyser les journaux systèmes.

Nous voyons à travers ces travaux que les défaillances peuvent être prédites efficacement grâce aux informations contenues dans les données de supervision. De plus, la plupart des travaux utilisent des méthodes d'apprentissage machine pour la prédiction. Les évaluations de ces travaux montrent que ces méthodes permettent d'atteindre de très bons résultats. Cependant, aucun des auteurs n'utilise la même méthode d'apprentissage machine pour réaliser les prédictions et n'utilise les mêmes données de supervision. De plus, l'évaluation de plusieurs méthodes d'apprentissage par certains auteurs montre qu'il n'existe pas de méthodes fournissant les meilleurs résultats possibles dans toutes les situations mais que chaque méthode présente des avantages selon les caractéristiques recherchées. Nous voyons donc que même si l'utilisation de méthodes d'apprentissage machine peut permettre d'atteindre de très bons résultats de prédiction en utilisant les données de supervision, la mise en place et le choix des méthodes n'est pas évident. Il dépend du cas d'étude et des caractéristiques recherchées.

Apports de la prédiction des défaillances dans les solutions de tolérance aux défaillances

Les résultats des travaux sur la prédiction des défaillances peuvent être utilisés pour améliorer les solutions actuelles de tolérance aux fautes. En effet, il peut être envisageable par exemple de réduire la fréquence de la sauvegarde des points de reprise des applications si nous connaissons avec précision la fréquence des défaillances. Ainsi, une sauvegarde de l'application ne serait réalisée que si le délai depuis la dernière défaillance devient trop

grand et que, par conséquent, la probabilité d'apparition d'une nouvelle défaillance est importante.

Bautista et al. [8] proposent une analyse des défaillances de plusieurs systèmes HPC utilisés en production incluant Titan et Blue Waters pour améliorer une solution de sauvegarde de points de reprise. L'utilisation des résultats de leur analyse permet de réduire jusqu'à 30% le coût de la solution de sauvegarde pour le système HPC étudié. De la même manière, Gainaru et al. [38] proposent une analyse des défaillances dans les systèmes HPC et évaluent la réduction du coût de la solution de sauvegarde des points de reprise de l'application dans le cas où elle serait complétée par une solution de prédiction des défaillances. Ils concluent qu'en considérant un temps moyen entre deux défaillances de 5 heures, une solution de prédiction des défaillances capable de prédire 65% des défaillances 10 secondes en avance avec une précision de 92% réduit le coût de la sauvegarde des points de reprise de l'application de 24,78%.

Ces deux travaux mettent en évidence que les solutions proposées autour de la prédiction des défaillances peuvent effectivement aider à améliorer les mécanismes de gestion des défaillances.

À travers cette section, nous avons vu que les défaillances sont fréquentes dans les systèmes HPC et qu'il existe déjà des solutions mises en place pour gérer ces défaillances. Ces solutions font principalement appel à la sauvegarde régulière de l'état courant de l'application. Lorsqu'une défaillance survient, le précédant état de l'application est rechargé pour continuer les calculs à partir de ce point de sauvegarde. Des travaux récents proposent d'augmenter l'efficacité de ces solutions en analysant et prédisant les défaillances. Cette analyse des défaillances utilise les données de supervision produites par les systèmes HPC. La plupart des solutions étudiées utilise l'apprentissage machine pour automatiser l'analyse des données de supervision et permettre de pouvoir détecter et prédire efficacement les défaillances à l'aide de cette analyse. Cependant, l'utilisation de ces méthodes n'est pas évidente et il n'existe pas une méthode universelle permettant d'obtenir les meilleurs résultats dans toutes les situations. Il est ainsi nécessaire de faire un choix en fonction des données analysées mais également en fonction des caractéristiques souhaitées de la solution. Dans la prochaine partie, nous allons décrire ces méthodes d'apprentissage machine et leurs propriétés. Nous expliquons ensuite leur utilisation pour automatiser l'analyse des données de supervision et pour prédire des défaillances.

2.3 Apprentissage machine

L'utilisation de l'apprentissage machine peut nous aider dans la tâche de construction d'une solution permettant d'automatiser l'analyse des données de supervision.

L'apprentissage machine est un domaine mêlant informatique et mathématiques. Il fournit un ensemble de méthodes permettant de calculer une fonction de transfert entre les entrées et les sorties que nous lui fournissons. En d'autres termes, il va nous permettre de trouver automatiquement un modèle faisant correspondre nos entrées et nos sorties. Nous commencerons cette partie par décrire l'utilisation de méthodes d'apprentissage machine pour l'automatisation de l'analyse des données de supervision. Puis, nous décrirons succinctement le fonctionnement des méthodes d'apprentissage machine en détaillant quelques modèles.

2.3.1 Utilisation de l'apprentissage machine dans le cadre de l'automatisation de l'analyse des données de supervision

Comme présenté précédemment, les systèmes HPC produisent une grande quantité de données de supervision. Ces données sont à la fois des données de type métrique correspondant à des valeurs numériques de capteur mais aussi des données de type textuel correspondant à des journaux systèmes. Ces données décrivent l'état du système.

Lorsqu'une défaillance survient dans le système, nous pouvons supposer que certaines de ces données de supervision peuvent la décrire. Par exemple, dans le cas d'une saturation sur le réseau d'interconnexion, nous pouvons supposer que la donnée de supervision décrivant la bande passante disponible du réseau diminue jusqu'à atteindre les capacités maximales du réseau. Dans ce cas, nous pouvons établir une règle simple sur cette donnée. Cette règle détectera une anomalie si la bande passante du réseau est inférieure à un seuil.

Cependant, ces simples règles risquent de devenir rapidement complexes car elles vont devoir tenir compte de la complexité à la fois des systèmes HPC mais aussi des données de supervision. En reprenant l'exemple de la saturation sur le réseau d'interconnexion, pour qu'un seuil soit réellement efficace, il est nécessaire de le placer à la bonne valeur pour rendre compte réellement d'une saturation du réseau et non pas d'une utilisation normale intensive. Doit-on considérer alors qu'il risque de survenir une défaillance si la bande passante disponible sur le réseau est en dessous de 10% ou plutôt en dessous de 5% ? Il peut être aussi intéressant de se focaliser sur la vitesse de la réduction de la banque passante disponible du réseau : si la bande passante disponible diminue rapidement il peut être intéressant d'avoir un seuil élevé pour détecter la défaillance avant qu'il ne soit trop tard, à l'inverse si elle diminue doucement un seuil élevé risque d'introduire de nombreuses fausses détections. Nous voyons rapidement qu'écrire l'ensemble des règles permettant de décrire les défaillances à partir des données de supervision peut être complexe et fastidieux.

L'utilisation de l'apprentissage machine peut aider à la *découverte* de règles automatiques à partir des données de supervision pour détecter et prédire les défaillances. En reprenant l'exemple précédant, nous pouvons fournir en entrée du modèle d'apprentissage machine la bande passante disponible du réseau et en sortie les cas de défaillance du réseau. Le modèle *apprendra* donc quelles sont les corrélations entre le *comportement* de la bande passante et les défaillances. Il nous permettra par la suite de détecter automatiquement les défaillances lorsque le *comportement* de la banque passante ressemblera à un comportement déjà vu menant une défaillance.

Il est à noter que l'exemple utilisé ici pour illustrer l'utilisation de l'apprentissage machine pour aider à la détection des défaillances est volontairement simpliste. Les défaillances peuvent être plus compliquées à détecter car elles peuvent dépendre de multiples facteurs.

Nous allons maintenant décrire comment fonctionne ces modèles d'apprentissage machine et comment nous pouvons les appliquer sur les données de supervision.

2.3.2 Description des modèles d'apprentissage machine

Les systèmes d'apprentissage machine sont divisés en deux catégories principales : les systèmes supervisés et les systèmes non supervisés [34]. La différence entre ces systèmes se situe au niveau de l'apprentissage. Dans le cas d'un système supervisé, il est entraîné en connaissant lors de son entraînement la sortie. Dans le cas d'un système non supervisé, il est entraîné en ne connaissant pas lors de son entraînement la sortie.

Reprenons l'exemple de la défaillance sur le réseau d'interconnexion. L'entrainement d'un système supervisé supposera de mettre en entrée du système la bande passante disponible du réseau et en sortie la défaillance détectée ainsi que la bande passante lorsqu'il n'y a pas de défaillance détectée. Le système apprendra alors précisément les différences entre le *comportement* de la bande passante lorsqu'il survient une défaillance et lors d'un comportement normal. Dans le cas d'un système non supervisé, l'entrée du système sera toujours composée de la bande passante disponible, mais nous ne spécifions pas au système à quel moment survient la défaillance. Il essayera alors de classer automatiquement les différents comportements possibles de la banque passante en fonction de ces propriétés : augmentation ou diminution rapide de la bande passante disponible, pic d'activité, taux de bande passante disponible, etc. Il décrira donc les données d'entrée en fonction de différents paramètres. Nous pourrons ensuite dans un deuxième temps désigner l'un ou plusieurs de ces paramètres comme indicateurs d'une défaillance.

Il est évident que les systèmes supervisés sont, la plupart du temps, les plus performants car nous leurs indiquons spécifiquement les défaillances ou les événements que nous souhaitons détecter [49, 55, 86, 53]. Cela leur permet de mettre en évidence plus facilement des paramètres spécifiques à prendre en compte pour détecter efficacement les défaillances par rapport aux systèmes non supervisés qui eux essaient de décrire au mieux l'ensemble des comportements possibles sans tenir compte de l'événement que nous souhaitons détecter.

Cependant, les systèmes supervisés demandent une annotation des données de supervision pour savoir précisément quand et où surviennent les défaillances afin de les fournir en sortie de ces systèmes. Il paraît peu envisageable de demander à un administrateur système d'annoter manuellement l'ensemble des données de supervision produites. Il peut donc être nécessaire de mettre en place une annotation automatisée des données. Cette annotation automatisée peut par exemple utiliser les informations contenues dans les journaux systèmes pour détecter les défaillances et ensuite annoter les métriques correspondantes à partir de ces informations. Cependant, dans des cas où l'automatisation de l'annotation n'est pas possible, l'utilisation des systèmes non supervisés est requise car ils ne requièrent pas d'annotations.

Nous voyons donc que les deux approches peuvent être utilisées selon le cas d'usage envisagé et selon les données dont nous disposons.

Nous allons maintenant décrire plus précisément la construction des modèles d'apprentissage machine. Le but de ce document n'étant pas de faire un état de l'art sur les différents modèles d'apprentissage machine, nous nous focalisons uniquement sur la description de quelques modèles classiques de l'état de l'art. De plus, dans la suite de ce document, nous utilisons uniquement des méthodes d'apprentissage supervisé car dans notre cas l'annotation des données peut se réaliser de manière automatisée à l'aide de l'analyse des journaux systèmes. Nous nous concentrons donc uniquement sur la description des modèles d'apprentissage supervisé.

2.3.3 Apprentissage supervisé

L'apprentissage supervisé est la méthode la plus connue du fait de l'essor récent des modèles d'apprentissage profond (*deep learning*) [54]. Cette méthode est réalisée en deux phases : la phase d'apprentissage et la phase d'inférence.

Lors de la phase d'apprentissage, le modèle optimise ses paramètres dans le but de faire correspondre l'entrée et la sortie que nous lui fournissons. Lors de la phase d'inférence, le modèle utilise les paramètres précédemment optimisés pour prédire la sortie à partir d'une

entrée fournie.

Le modèle peut apprendre deux types de sorties : continue (modèle de *régression*) ou discrète (modèle de *classification*).

Lors d'une classification, les différentes classes que représentent la sortie sont des catégories. C'est-à-dire qu'elles n'ont pas de liens entre elles et qu'on ne peut pas calculer directement une distance entre elles. Prenons par exemple la classe *chien* et la classe *chat* qui représentent deux classes différentes. Il est difficile de calculer une distance entre la classe *chien* et la classe *chat*. Dans le cas d'une régression, la sortie peut prendre un ensemble continu de valeurs. Par exemple dans le cas de la prédiction de température, la température peut être n'importe quelle valeur. On peut facilement calculer des distances entre les valeurs. La distance entre la température 4°C et la température 14°C est de 10°C.

Ces deux types de sorties entraînent des différences dans la façon dont les modèles d'apprentissage machine fonctionnent. Dans le cas d'une classification, la sortie du modèle sera un vecteur de la taille du nombre de classes et la classe sélectionnée sera celle avec la plus haute valeur dans le vecteur. Dans le cas d'une régression, la sortie sera directement la valeur attendue.

Nous allons maintenant décrire de manière plus précise deux grandes classes de modèles d'apprentissage supervisé : les modèles d'apprentissage profond et les modèles basés sur des arbres.

Modèle d'apprentissage profond

Les modèles d'apprentissage profond représentent actuellement l'état dans l'art dans des domaines variés tel que la classification d'images [98], la détection de personnes [32] ou encore l'analyse de série temporelle [39, 96]. Ces modèles utilisent l'assemblage de briques élémentaires : des neurones. La figure 2.3 présente un neurone. Celui-ci est composé de 3 parties : les entrées, la fonction d'activation et la sortie. Dans un premier temps, les entrées sont multipliées par des poids. La somme de ces entrées pondérées est ensuite réalisée. Puis cette somme est utilisée par une fonction d'activation qui détermine la valeur de sortie du neurone. Cette fonction d'activation peut être simplement un seuil. Dans le cas de l'utilisation d'un seuil comme fonction d'activation, si la somme des entrées est supérieure à ce seuil, la valeur de la sortie sera de 1, sinon elle sera de 0.

Pour créer un réseau de neurones, il est nécessaire d'empiler les neurones en couches. La figure 2.4 présente un réseau de neurones à 5 couches. La première couche est appelée couche d'entrée, les deux suivantes sont appelées couche cachée et la dernière est appelée couche de sortie. Nous avons donc ici 3 neurones pour la couche d'entrée, 3 neurones par couche cachée et un neurone pour la couche de sortie. Chaque sortie d'un neurone est connectée à l'entrée d'un autre neurone. En ajustant les poids de chaque entrée, le réseau peut *apprendre* à faire correspondre les valeurs en entrée avec la sortie souhaitée. L'ajustement des poids se fait lors de la phase d'apprentissage en se basant sur la rétro-propagation du gradient. L'idée de la rétro-propagation du gradient est de calculer la différence entre la sortie obtenue et la sortie voulue, puis d'ajuster les poids dans le réseau afin de faire correspondre la sortie obtenue et la sortie désirée. La rétro-propagation du gradient requiert des propriétés mathématiques sur la fonction d'activation telles que la différentiabilité. Nous ne décrirons pas ici l'algorithme de rétro-propagation du gradient car cela est en dehors du périmètre de cette explication des modèles d'apprentissage machine. Cependant, le lecteur pourra trouver dans la littérature l'explication complète de celui-ci [47].

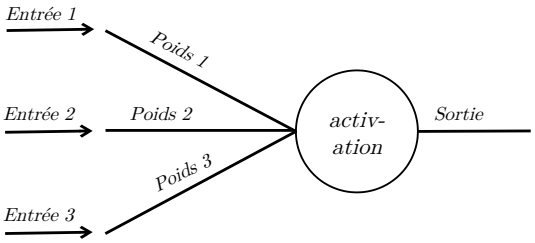


FIGURE 2.3 – Présentation d'un neurone.

La manière d'empiler les neurones et de les connecter entre eux permet de spécialiser des couches dans le but de spécialiser le réseau dans certains types de tâches comme l'analyse d'image ou l'analyse des signaux temporels. Dans la grande majorité des cas, l'architecture d'un réseau de neurones est composée de plusieurs types de couches différentes qui se découpent en deux grandes parties : une première partie responsable de l'extraction des caractéristiques intéressantes des données et une deuxième partie responsable de la classification de ces caractéristiques.

La figure 2.5 présente trois types de réseau de neurones différents : la figure 2.5a montre un réseau pleinement connecté, la figure 2.5b montre un réseau convolutionnel et la figure 2.5c montre un réseau récurrent. Chaque type de réseau dispose de caractéristiques propres.

Le réseau convolutionnel n'est connecté qu'à une partie des neurones de la couche suivante ce qui le destine à l'extraction des caractéristiques d'image dans des zones précises. Chaque neurone s'occupe alors d'une partie de l'image, chaque nouvelle couche ayant accès à une vue plus globale de l'image.

Le réseau récurrent est connecté non seulement entre les couches mais aussi entre les neurones d'une même couche. Cela permet d'échanger, dans le cadre d'analyse des signaux temporels, de l'information temporelle entre les neurones d'une même couche. En effet, si l'on considère que l'entrée 1 est le premier point du signal, puis l'entrée 2 le deuxième point et ainsi de suite, l'information peut être transmise de l'entrée 1 à l'entrée 3 à l'intérieur de la première couche. Ce type de réseau se destine plus particulièrement à l'analyse des signaux temporels.

Enfin, le réseau pleinement connecté est le réseau de base utilisé très souvent après des couches spécialisées. Il est responsable de la classification des caractéristiques extraites par les couches spécialisées.

Bien que les méthodes utilisant des modèles d'apprentissage profond représentent l'état de l'art dans beaucoup de domaines, elles nécessitent de très grandes quantités de données pour être utilisables [17, 83]. De plus, ce sont des modèles *boîtes noires*. Il est complexe et difficile de pouvoir interpréter la connaissance apprise par le réseau à la fin de l'étape d'apprentissage [104]. En d'autres termes, il est complexe de déterminer sur quel élément de nos données d'entrée le réseau s'est basé pour prédire la sortie. À l'inverse de ces méthodes *boîtes noires* et qui nécessite une grande quantité de données, il existe des méthodes adaptées à des petites quantités de données et qui peuvent être facilement interprétables. Ces méthodes utilisent des arbres. Nous les présentons dans la prochaine partie.

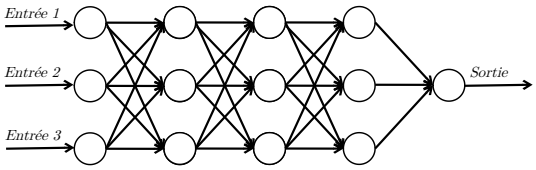
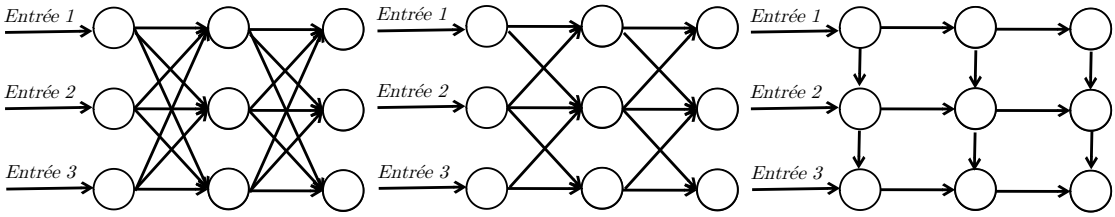


FIGURE 2.4 – Présentation d'un réseau de neurones.



(a) Réseau de type pleinement connecté (b) Réseau de type convolutionnel (c) Réseau de type récurrent

FIGURE 2.5 – Exemple de différents types de réseau de neurones.

Modèles d'apprentissage basés sur des arbres

Bien que les modèles d'apprentissage profond représentent actuellement l'état de l'art dans beaucoup de domaines, ils ont besoin de grandes quantités de données pour être performants. Ainsi, dans le cas où l'on dispose de peu de données, il est intéressant de travailler avec des modèles d'apprentissage classiques. Comme précisé précédemment, notre but n'étant pas de faire un état de l'art complet des méthodes d'apprentissage machine, nous allons nous concentrer sur quelques méthodes. Ces méthodes sont basées sur la construction d'arbres. Nous décrirons ici 3 méthodes différentes : arbre décisionnel [82], forêts aléatoires [12] et *Gradient Boosting Tree* [35]. Chacune de ces méthodes a des propriétés différentes que nous allons présenter. Il est à noter que ces méthodes peuvent être aussi bien utilisées pour réaliser de la classification ou de la régression. Les exemples suivants illustrent uniquement l'utilisation de ces méthodes pour réaliser une tâche de classification.

Les arbres décisionnels sont des arbres où les feuilles de l'arbre représentent les classes de notre jeu de données. Chaque branche est une condition sur les données d'entrée permettant de cheminer dans l'arbre jusqu'aux feuilles. La figure 2.6 présente un arbre de décision permettant de déterminer si le processeur est dans un état de surchauffe ou non. Les deux états sont décrits par des ovales et représentent les feuilles de l'arbre. Les conditions sont décrites par des rectangles. La branche droite des décisions indique une condition vraie, alors que la branche gauche indique une condition fausse. Dans cet exemple, l'arbre de décision se base sur trois données différentes pour déterminer l'état du processeur : la température de celui-ci, l'activité réseau et l'activité du processeur. L'exemple donné permet de dire, par exemple, que si la température du processeur est supérieure à 60°C et que son activité est supérieure à 80% alors le processeur est dans un état de surchauffe. Le principal avantage des arbres décisionnels est leur interprétabilité. Ce sont des modèles *boîtes blanches*. Cependant, ils peuvent rapidement devenir larges et profonds du fait de leur simplicité.

La méthode des forêts aléatoires propose de construire plusieurs arbres de taille moyenne au lieu de construire un seul grand arbre décisionnel. L'apprentissage se fait en sélectionnant de manière aléatoire des exemples dans les données d'apprentissage. Chaque sélection sera

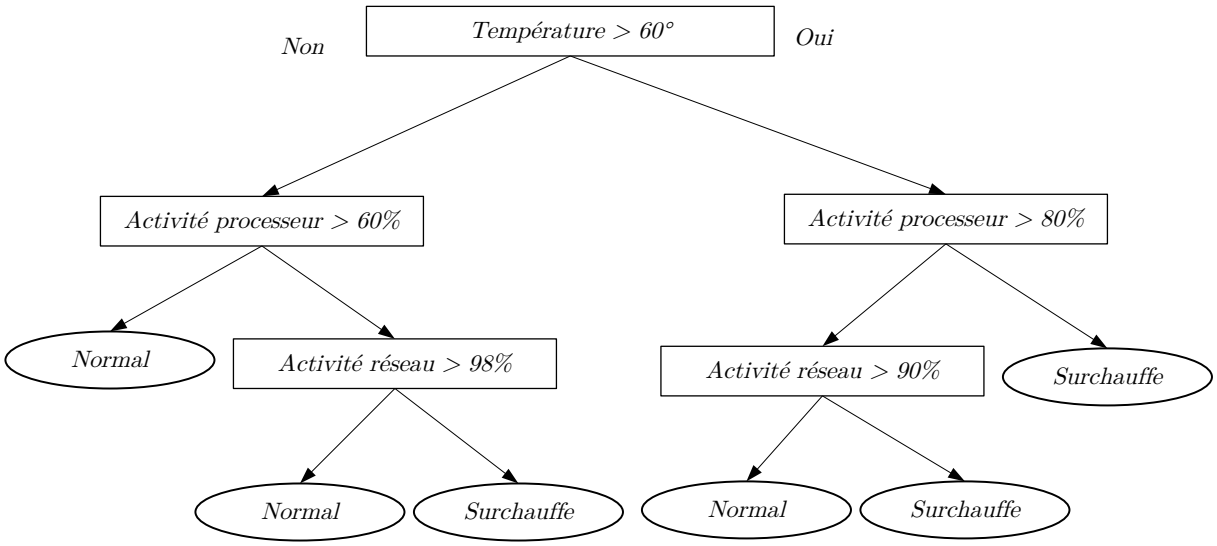


FIGURE 2.6 – Exemple d'un arbre de décision utilisant l'activité réseau, l'activité processeur et la température du processeur pour déterminer si le processeur est dans un état de surchauffe.

ensuite utilisée pour construire un arbre de décision. Une fois l'apprentissage terminé, la prédiction se fait en utilisant un vote majoritaire sur l'ensemble des arbres appris. Cette construction permet d'éviter le sur-apprentissage. Le sur-apprentissage est l'apprentissage *par cœur* des données d'apprentissage en perdant la capacité de généralisation sur les données de test. En construisant plusieurs arbres avec pour chacun une sous-sélection des données d'apprentissage, aucun arbre n'utilise l'ensemble des données d'apprentissage. Et par conséquent, aucun arbre n'est capable d'apprendre *par cœur* l'ensemble des données d'apprentissage.

Enfin, le *Gradient Boosting Tree* étend la notion d'arbre en utilisant un gradient de valeur au lieu d'utiliser directement les valeurs d'entrées. Le principe est de construire un premier petit arbre. Puis, un second arbre est construit à partir de la classification fournie par le premier arbre et de la différence entre la classification attendue et la classification réelle. La construction d'un ensemble de *classifieurs* (technique appelée *Boosting*) permet d'augmenter les performances de généralisation de l'algorithme car la technique construit un ensemble de *petits* arbres. Chacun est en charge d'une classification faible des données (*weak classifiers*).

Le choix de l'approche dépend des données utilisées en entrée, mais aussi des propriétés attendues du système. En effet, chacune de ces approches à des propriétés différentes : les arbres de décision sont facilement interprétables mais leur simplicité peut limiter leur capacité d'apprentissage et de généralisation, les forêts aléatoires proposent une approche légèrement différente permettant de résoudre le problème de généralisation, enfin les Gradient Boosting Tree proposent une approche utilisant les résidus pour augmenter encore les capacités d'apprentissage mais deviennent complexes à interpréter.

Nous voyons à travers cette section qu'il existe un nombre important de méthodes d'apprentissage machine et que chacune présente des caractéristiques différentes. Le choix de la méthode n'est donc pas évident de par leurs différentes caractéristiques mais également de par le type et la volumétrie des données d'entrée envisagées. La sélection de la bonne méthode nécessite donc un travail poussé d'analyse des données et des propriétés attendues.

2.4 Synthèse

La précédente description d'un système HPC permet de comprendre qu'un système HPC est un ensemble complexe de composants s'interconnectant pour offrir la meilleure performance. Cependant, des défaillances peuvent survenir dans ces systèmes et en réduire l'efficacité. Des solutions sont actuellement mises en place pour tenter de réduire le coût de ces défaillances sur le système. Elles reposent principalement sur l'hypothèse que les défaillances surviennent mais qu'il est nécessaire de mettre en place des actions pour permettre aux applications de continuer s'exécuter en dépit des défaillances.

Actuellement, la solution la plus couramment employée fait appel à l'utilisation de points de reprise des applications. Si une défaillance survient lors de l'exécution de l'application et interrompt son exécution, il suffit de restaurer l'état de l'application depuis le dernier point de reprise. Cependant, cette solution est coûteuse. Elle nécessite un espace de stockage important pour enregistrer l'état de l'application, elle impose des transferts de données important sur le réseau, elle perturbe le fonctionnement de l'application lors de la sauvegarde, etc.

Des travaux récents tentent de réduire le coût de ces solutions de sauvegarde de points de reprise ou proposent des solutions alternatives dans le but de réduire le coût des défaillances sur le système. Des travaux utilisent notamment l'analyse des données de supervision pour mieux comprendre les défaillances et être capable, dans le meilleur des cas, de les prédire. Cependant, l'analyse ces données de supervision n'est pas facile à réaliser.

Le premier défi est lié à la volumétrie et diversité des données de supervision. En effet, les données de supervision sont issues de l'ensemble des composants du système et reflètent donc la complexité d'un système HPC. Elles sont de différents types (numérique ou textuel principalement) et représentent un très grand volume de données. Dans le système HPC étudié dans ce manuscrit présenté dans le paragraphe 3.1, plus de 100 000 entrées de données textuelles sont écrites par minute et plus de 7 millions de points de valeur concernant les données numériques sont collectés par minute.

De plus, l'interconnexion de l'ensemble des composants entraîne une analyse des défaillances complexes. En effet, une défaillance sur le réseau d'interconnexion peut empêcher une application d'accéder à des fichiers situés sur le système de stockage. Ainsi, pour pouvoir corriger efficacement la défaillance, il sera nécessaire de mettre en évidence que le dysfonctionnement de l'application est dû à un problème d'accès à un fichier sur le système de stockage et que ce problème est lui-même dû à une défaillance sur le réseau d'interconnexion. Cette réaction en chaîne illustre le besoin de connaissances pointues sur le système par un opérateur humain pour permettre un diagnostic correct de la défaillance. Cependant, au vu de la volumétrie des données de supervision et de la fréquence des défaillances une analyse manuelle est fastidieuse.

Le deuxième défi est donc lié au choix des méthodes pour réaliser cette analyse de manière automatisée. Les solutions proposées pour l'analyse des données de supervision mettent en œuvre fréquemment des modèles d'apprentissage machine pour aider à l'automatisation de cette analyse. Néanmoins, le choix de la méthode d'apprentissage machine est une étape cruciale mais compliquée pour obtenir des corrélations fiables. Ce choix dépendra à la fois du type mais aussi de la volumétrie des données et des défaillances.

Enfin, le dernier défi correspond au but envisagé. Dans le cas de la prédiction d'une défaillance, l'analyse doit être suffisamment rapide pour envisager de prendre des actions pouvant mitiger le coût de ces défaillances. De la même façon, les résultats de l'analyse

doivent être fiables pour ne pas générer un nombre important de faux positifs. Ces faux positifs peuvent réduire l'efficacité de la solution si des actions pour corriger les défaillances sont entreprises alors qu'aucune défaillance n'allait survenir. Dans le cas où le but recherché est de mieux comprendre les défaillances en mettant en évidence les relations entre les différents événements survenant dans un système HPC, il est nécessaire d'avoir une analyse qui soit capable de mettre en évidence les relations pour la plupart des événements survenant dans un système HPC. En effet, pour que l'analyse soit utile, elle doit pouvoir détecter les relations de n'importe quel événement survenant dans le système. De plus, de la même manière que pour la prédiction, elle doit être assez rapide pour être utilisée par un utilisateur qui souhaite mieux comprendre une défaillance.

L'ensemble de ces défis et des contraintes associées montrent que l'automatisation de l'analyse des données de supervision est difficile et complexe à mettre en œuvre.

Les contributions de cette thèse s'inscrivent dans les travaux qui s'articulent autour de l'automatisation de l'analyse des données de supervision. Nous proposons deux contributions principales avec deux objectifs différents. La première contribution vise la prédiction d'une défaillance particulière qui est la surchauffe du processeur. Dans le chapitre 3, nous proposons une analyse détaillée de ce type de défaillance et dans le chapitre 4 nous détaillons la solution développée pour prédire ces surchauffes. La deuxième contribution vise une meilleure compréhension des défaillances en général. Ainsi, dans le chapitre 5, nous proposons une solution permettant d'aider au diagnostic générique des défaillances en se basant sur l'analyse des journaux systèmes.

Chapitre 3

Analyse des événements de surchauffe dans un système HPC

Dans le chapitre 2, nous avons montré que des défaillances peuvent survenir fréquemment dans les systèmes HPC et qu'elles peuvent réduire la disponibilité de ces systèmes. Nous proposons dans ce chapitre l'étude d'une défaillance survenant les systèmes HPC : la surchauffe des processeurs. La surchauffe d'un processeur est un événement qui se produit lorsque le processeur dépasse un seuil de température. À la suite de ce dépassement et afin d'éviter des dommages matériels, des actions sont entreprises par le processeur afin de faire baisser sa température.

Nous nous intéressons dans ce chapitre à l'étude des causes et des conséquences de la surchauffe des processeurs dans les systèmes HPC. Les conclusions de ce chapitre nous aideront dans le chapitre 4 à construire un système permettant de prédire ces surchauffes. En effet, l'identification des causes nous aidera à nous focaliser sur les données importantes permettant de prédire efficacement les surchauffes et la détermination des conséquences nous permettra d'estimer le coût de celles-ci et de déterminer des actions préventives afin de mitiger leur influence sur le système.

Nous commençons ce chapitre par la section 3.1 qui détaille le système HPC et les données dont nous disposons pour réaliser les études présentées dans ce manuscrit. Nous abordons ensuite les problématiques liées à la recherche des causes et conséquences des surchauffes dans un système HPC dans la section 3.2. Puis, nous définissons précisément un événement de surchauffe du processeur et les informations disponibles pour le détecter dans la section 3.3. Ensuite, nous proposons une analyse des causes de la surchauffe en détaillant les corrélations possibles avec les données dont nous disposons dans la section 3.4. Enfin, nous détaillons les conséquences des surchauffes sur le système et leur influence sur la disponibilité du système dans la section 3.5.

3.1 Système étudié

Les travaux que nous présentons par la suite utilisent les données collectées sur le système HPC de DKRZ [1]. Ce système est composé de deux types de nœuds : 1550 nœuds équipés de 2 processeurs Intel Xeon E5-2680 V3 de 12 cœurs et de 64 Go de mémoire vive et 1750 nœuds équipés de 2 processeurs Intel Xeon E5-2695 V4 de 18 cœurs et de 64 Go de mémoire vive. Il est accompagné de 21 nœuds dédiés à la visualisation de données

1. Centre allemand de climatologie https ://www.dkrz.de/

scientifiques équipés de cartes graphique fournies par Nvidia. Le stockage est confié au système de fichiers distribué Lustre pour une capacité totale de 54 Po et une bande passante maximale de 450 Go/s. Le réseau haute-performance utilise la technologie InfiniBand FDR interconnect avec une bande passante de 6 Go/s par nœud et des commutateurs réseaux Mellanox SX6536. Son pic de performance est de 3.6 PetaFlops (ce qui équivaut à 3.6 millions de milliards d'opérations par seconde). Enfin son refroidissement est assuré par un système de refroidissement à eau tiède. Ce système diffère des systèmes classiques de refroidissement utilisant une climatisation car il repose sur l'utilisation d'eau à température ambiante et d'un couple radiateur/ventilateur classique pour refroidir l'eau. Il ne fait donc pas appel à un système de climatisation pour refroidir l'air ou l'eau ce qui permet de réduire la consommation énergétique.

Les données issues de ce système se découpent en 3 grandes parties : les journaux systèmes, les métriques et la base de données du gestionnaire de ressources.

Les journaux systèmes représentent les entrées (*system logs*) écrites par le système d'exploitation (*Red Hat*) des nœuds de calcul. Ils sont découpés par fichier. Chaque fichier représente les entrées correspondant à un nœud de calcul pour un mois. Les journaux systèmes sont disponibles de mars 2017 à décembre 2018. Ils sont composés d'environ 1 milliard d'entrées pour un poids d'environ 130 Go.

Les métriques sont collectées à travers l'ensemble des composants du système. Elles représentent l'évolution de 2505 indicateurs comme la température des processeurs, la bande passante réseau, la mémoire vive disponible, etc. Elles sont enregistrées dans une base de données OpenTSBD [2] qui est spécialisée dans l'enregistrement et le traitement des séries temporelles. Les métriques sont disponibles de mars à septembre 2017 et représentent 1,2 To de données.

La base de données du gestionnaire de ressources comprend les données de mars 2017 à mai 2018. Elle est composée de l'heure de début, de fin, de la durée et des ressources réservées de toutes les applications qui s'exécutent sur le système. Elle comprend environ 5 millions d'exécutions d'application.

3.2 Problématique

La surchauffe des processeurs dans les systèmes HPC est un événement se produisant lorsque la température du processeur dépasse un seuil fixé par le constructeur de ce processeur. Pour éviter des dommages matériels, lorsque la surchauffe survient, le processeur baisse progressivement sa consommation énergétique afin de faire baisser sa température. Cette baisse de consommation induit une baisse de la puissance de calcul du processeur. Nous commençons cette section par donner une vue d'ensemble des surchauffes survenant dans différents types de systèmes. Puis nous nous focalisons sur les travaux traitant des causes et des conséquences des surchauffes.

3.2.1 Vue d'ensemble des surchauffes

Une surchauffe est un événement arrivant lorsque la température du composant dépasse un seuil prédéfini. Elle peut survenir sur plusieurs types de composants : processeurs, cartes graphiques, disques durs, etc.

2. http ://opentsdb.net/

L'influence d'une haute température sur les composants peut être multiple : augmentation des erreurs générées par le composant [70], réduction de la fiabilité [81, 19] et de la durée de vie [91], etc. La gestion de la température est donc un enjeu majeur pour assurer la fiabilité de ces composants et des systèmes qui les utilisent. Cependant, la gestion de la température par des systèmes de refroidissement est responsable d'une part importante des coûts énergétiques d'un système [78].

Dans ce contexte, un compromis doit être trouvé entre le refroidissement efficace des composants pour assurer leur fiabilité et la minimisation du coût de refroidissement de ces composants. Pour parvenir au meilleur compromis, il est donc nécessaire de dimensionner de manière idéale le refroidissement du système pour minimiser le coût de celui-ci, tout en évitant au maximum les surchauffes des composants. Des solutions sont donc développées pour réduire ce coût comme le refroidissement utilisant de l'eau chaude [51].

Cependant, malgré ces solutions, des surchauffes peuvent survenir dans le système lorsque certaines conditions sont réunies. Bien que l'influence d'une haute température sur les composants soit étudiée dans l'état de l'art, la surchauffe des processeurs dans les systèmes HPC n'est, à notre connaissance, pas étudiée. Nous proposons donc dans ce chapitre une étude des causes et conséquences des surchauffes des processeurs dans un système HPC. L'étude de ce type de défaillance requiert d'analyser les causes menant à cet événement mais également ces conséquences sur le processeur et sur le système.

3.2.2 Causes d'une surchauffe dans un système HPC

La cause directe de la surchauffe est une augmentation de la température. Cependant, il est utile de connaître dans quelles conditions cette montée en température s'opère et si nous sommes capables de déterminer si des conditions d'apparitions précises des surchauffes peuvent être mises en évidence.

De manière plus générale, il existe des travaux qui traitent des corrélations entre les défaillances dans les systèmes HPC et certains paramètres. Les travaux de Gupta et al. [40] et Wang et al. [94] mettent en évidence des corrélations entre les défaillances survenant sur les systèmes HPC et la période de l'année. El et al. [29] mettent eux en évidence des corrélations entre les défaillances et la charge du système. Cependant, aucun de ces travaux n'intègre la surchauffe en tant que défaillance.

Nos travaux vont donc se concentrer sur la recherche des corrélations possibles entre les événements de surchauffe et les caractéristiques du système. La section 3.4 détaille l'ensemble des analyses des données réalisées pour déterminer les causes de la surchauffe.

3.2.3 Conséquences d'une surchauffe

Lorsque la température d'un processeur augmente et qu'il entre dans un état de surchauffe, il entreprend deux actions dans le but de faire baisser sa consommation électrique et donc sa dissipation d'énergie. La première est une modulation de l'horloge (*clock modulation*), la deuxième est un ajustement de la fréquence d'horloge et de la tension électrique de fonctionnement. L'ajustement de la fréquence et de la tension impliquent que le processeur réduit sa fréquence de fonctionnement jusqu'à la fréquence minimale disponible par paliers. Ainsi, il est important de pouvoir déterminer quelle est l'influence exacte de la baisse de fréquence du processeur dans les applications distribuées exécutées par les systèmes HPC.

La première conséquence directe est donc une baisse de la fréquence de fonctionnement

du processeur. L'influence de cette baisse de fréquence du processeur sur les applications HPC est largement étudiée dans l'état de l'art dans le contexte de la réduction de la consommation énergétique des processeurs. Le but de ces travaux est de pouvoir baisser la fréquence des processeurs pour réduire leur consommation énergétique en influençant au minimum les applications s'exécutant sur ceux-ci. Ainsi, ils établissent une corrélation entre la baisse de la consommation énergétique du processeur (via une baisse de fréquence de celui-ci) et une augmentation du temps d'exécution de l'application s'exécutant sur celui-ci. Rountree et al. [77] montrent qu'il est possible de réduire la consommation d'énergie du processeur d'environ 9% sur les applications parallèles provenant du jeu d'applications NAS en augmentant le temps d'exécution de seulement 1%. D'autres travaux arrivent à des conclusions similaires [61, 76]. Cependant, ces travaux se concentrent sur des baisses de fréquence relativement petites (de l'ordre d'une dizaine de pourcents maximum) car leur but est d'influer le moins possible sur le temps d'exécution de l'application. Dans le cas de la surchauffe, la baisse de fréquence peut être importante (plus de 50%) [3]. Certains travaux se focalisent sur l'analyse de l'influence d'une baisse importante de fréquence. Lively et al. [58] montrent que sur l'application GTC, une baisse de la fréquence du processeur de 1.8Ghz à 1Ghz (soit une baisse de 45%) induit une augmentation du temps d'exécution de 2324 secondes à 3559 secondes (soit une augmentation de 53%). D'autres travaux arrivent à des conclusions similaires [5, 52]. Nous voyons donc ici qu'une baisse importante de fréquence peut très fortement augmenter le temps d'exécution de l'application. Cependant, ces travaux étudiant des baisses importantes de fréquence utilisent des applications peu distribuées. En effet, les études sont faites en utilisant une application fonctionnant sur un seul nœud ce qui ne représente pas les applications fortement distribuées principalement utilisées dans les systèmes HPC. Or, il paraît important de pouvoir quantifier l'importance de la distribution des applications sur l'influence de la baisse de fréquence d'un processeur utilisé par l'application.

Nos travaux seront donc dirigés vers la mise en évidence de l'influence d'une baisse de fréquence importante d'un processeur dans le cas des applications distribuées utilisées sur les systèmes HPC.

3.3 Identification et évaluation de la surchauffe

La première étape pour être capable de comprendre et déterminer les causes et les conséquences d'une surchauffe est d'identifier l'occurrence de ces événements dans le système HPC. Dans un premier temps, nous présentons les données utilisées pour détecter les événements de surchauffe, puis à l'aide de ces données nous proposons une définition de ceux-ci.

3.3.1 Identification de la surchauffe

L'identification des événements de surchauffe est réalisée par l'analyse des journaux systèmes.

Lorsqu'un processeur surchauffe, il lève une exception. Cette exception est capturée par le noyau Linux et une entrée est écrite dans le journal système. Afin de garder un nombre raisonnable d'entrées dans les journaux systèmes, le choix est fait dans le noyau Linux de

3. https ://www.intel.com/content/dam/www/public/us/en/documents/datas heets/xeon-e5-1600-2600-vol-1-datasheet.pdf

ne générer qu'une entrée toutes les 5 minutes [4] et non pas à chaque interruption levée par le processeur. Lorsque la première interruption est levée, le noyau écrit la première entrée. Puis, si dans les 5 minutes suivant la première entrée il y a eu au moins une nouvelle d'interruption levée depuis la dernière entrée, alors une nouvelle entrée est écrite.

Deux types d'entrée peuvent être écrits. Le premier type correspond à une surchauffe du processeur entier, ce qui implique que l'ensemble des cœurs du processeur subit la surchauffe. Cette surchauffe est une surchauffe de type *package-level*. Le deuxième type correspond à une surchauffe impliquant qu'un seul cœur du processeur. La fréquence de fonctionnement sera réduite uniquement sur le cœur en surchauffe, les autres cœurs du CPU fonctionnant à leur fréquence nominale. Cette surchauffe est une surchauffe de type *core-level*. Dans la suite de nos travaux, nous ne considérons que les surchauffes de type *package-level* car les données représentant la température des processeurs dont nous disposons ne décrivent que la température du *package* du processeur et non celle de chacun des cœurs. Il est à noter que dans la pratique, les deux types de surchauffes sont redondants. Nous n'avons jamais observé d'entrée correspondant à une surchauffe d'un seul cœur d'un processeur. En effet, lorsqu'une surchauffe de type *package-level* se produit, l'ensemble des cœurs génère en même temps une surchauffe de type *core-level*.

Les surchauffes de type *package-level* sont décrites par un message tel que le suivant :

```
CPU46: Package temperature above threshold, cpu clock throttled (total events = 23193)
```

À ce message s'ajoute des méta-informations. Ces méta-informations sont composées du nœud sur lequel la surchauffe survient et de l'heure précise de l'événement de surchauffe. Ainsi, en se basant sur cette entrée nous pouvons savoir précisément où et quand l'événement de surchauffe survient. Cependant, un événement de surchauffe n'est pas un événement ponctuel, il dure un certain temps et il nous est donc nécessaire de savoir comment déterminer le début et la fin d'un tel événement dans le système. Nous décrivons dans le prochain paragraphe les méthodes pour déterminer le début et la fin d'un événement de surchauffe en se basant sur l'analyse des journaux systèmes.

3.3.2 Définition et notation d'un événement de surchauffe

Nous avons vu qu'un événement de surchauffe est un événement ayant une durée variable et qu'il est décrit par des entrées dans les journaux systèmes. Cependant, ces entrées ne décrivent pas parfaitement l'événement. En effet, comme nous l'avons expliqué, une entrée est générée toutes les 5 minutes si une nouvelle interruption a été levée. Il est donc nécessaire de préciser quelles sont les entrées liées au début et à la fin de ces événements. En effet, si nous souhaitons analyser les causes d'apparition d'un événement de surchauffe, il est important de se focaliser sur les événements présents *avant* l'événement de surchauffe et non sur les événements présents *pendant* l'événement de surchauffe.

La figure 3.1 montre la caractérisation des différents types d'entrées que nous considérons :

Définition 1. *E_o décrit un événement de surchauffe.*

Définition 2. *L_o décrit une entrée correspondant à un événement de surchauffe dans les journaux systèmes.*

4. https ://elixir.bootlin.com/linux/latest/source/arch/x86/kernel/cpu/mcheck/therm_throt.c

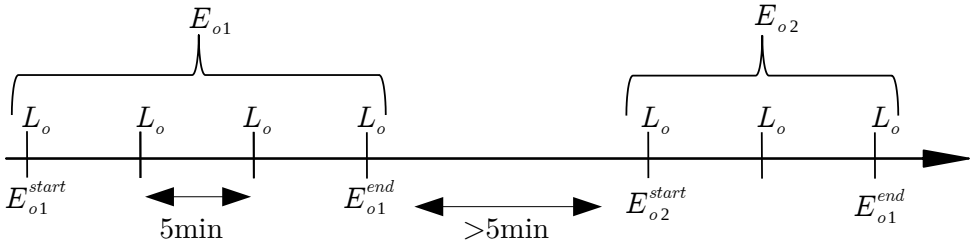


FIGURE 3.1 – Relation entre les entrées dans les journaux systèmes et les événements de surchauffe.

Définition 3. *E_o^{start} décrit le début d'un événement de surchauffe. Le début correspond à l'entrée L_o ayant une distance dans le temps de plus de 5 minutes avec la précédente entrée concernant le même processeur.*

Définition 4. *E_o^{end} décrit la fin d'un événement de surchauffe. La fin correspond à l'entrée L_o ayant une distance dans le temps de plus de 5 minutes avec l'entrée suivante concernant le même processeur.*

Ainsi, la figure 3.1 décrit 2 événements de surchauffe. Le premier est composé de 4 entrées dans les journaux systèmes et le deuxième est composé de 3 entrées.

Dans la suite de ce manuscrit, l'emploi des mots *événement de surchauffe* ou *surchauffe* référera uniquement aux entrées L_o qui correspondent au début des événements de surchauffe E_o^{start}.

3.4 Analyse des causes de la surchauffe

L'analyse des causes de la surchauffe doit nous permettre de mettre en évidence les corrélations entre les événements de surchauffe et les caractéristiques du système. Dans un premier temps, nous analysons les corrélations entre la charge du système et l'occurrence des événements de surchauffe, puis nous analysons les corrélations avec les applications fonctionnant sur le système, enfin nous terminons par analyser les corrélations avec la température des processeurs.

Il est à noter que plusieurs autres analyses pourraient être intéressantes à réaliser (corrélations avec des bibliothèques utilisées par les applications par exemple). Cependant, les analyses que nous pouvons effectuer sont limitées par les données de supervision collectées sur notre système. Pour les analyses suivantes, nous utilisons les données de température des processeurs, la base de données du gestionnaire de ressources et les journaux systèmes. Les données de température utilisées correspondent à la période de mars à septembre 2017. La base de données du gestionnaire de ressources et les journaux systèmes correspondent à la période de mars 2017 à mai 2018.

3.4.1 Charge du système

Dans un premier temps, nous analysons les corrélations avec la charge du système. Notre but est de savoir par exemple si une forte charge instantanée peut générer des événements de surchauffe.

En utilisant la base de données du gestionnaire de ressources, nous pouvons savoir quand un processeur ou plusieurs processeurs sont réservés par un utilisateur, et par

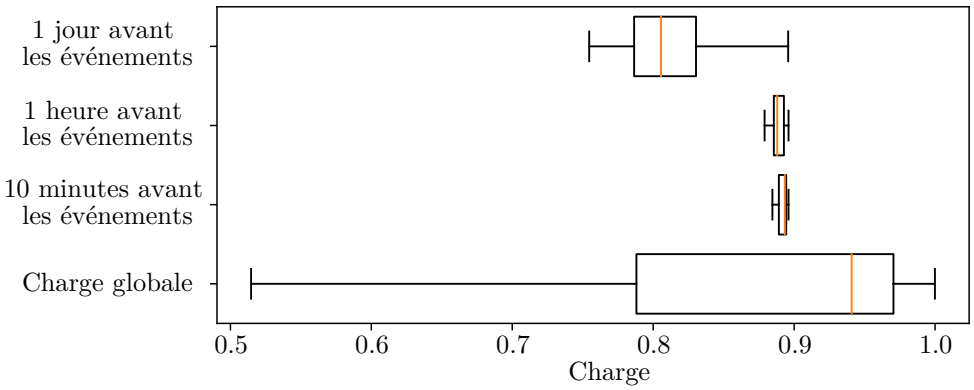


FIGURE 3.2 – Distribution de la charge globale du système et de la charge avant les événements de surchauffe.

conséquent déterminer la charge du système. Elle est définie par le nombre de ressources utilisées en même temps. Dans notre cas d'étude, les ressources étudiées se limitent aux processeurs car nous nous focalisons sur la surchauffe des processeurs.

La charge est donc définie par le nombre de processeurs réservés divisé par le nombre de processeurs total par unité de temps. Nous considérons que si un processeur est réservé tout ou en partie durant une unité de temps, alors il est réservé pour l'unité de temps entière. Par exemple, en considérant une unité de temps d'une heure, si un processeur est réservé de 14h35 à 15h45 nous considérons qu'il est réservé pour deux unités de temps (2 heures ici). Nous considérons aussi que lorsqu'un utilisateur réserve des ressources, celles-ci sont utilisées pendant toute la durée de la réservation.

La figure 3.2 montre la distribution de la charge du système durant les périodes analysées avec une unité de temps de 10 minutes et la distribution de la charge du système avant les événements de surchauffe (respectivement 10 minutes, 1 heure et 1 jour avant les événements). Pour rappel, sur la figure une charge de 0.8 signifie que durant une unité de temps de 10 minutes, 80% des processeurs du système HPC sont réservés par un ou plusieurs utilisateurs. Nous pouvons voir que les événements de surchauffe surviennent lorsque la charge du système est inférieure à la charge médiane globale (0.95). De même, nous ne voyons pas une augmentation nette de la charge entre l'heure (0.89) et les 10 minutes (0.9) avant les événements de surchauffe. Nous pouvons conclure que dans notre cas d'étude les événements de surchauffe ne surviennent pas lors d'une forte charge instantanée du système.

Cependant, bien que les surchauffes ne semblent pas être corrélées avec une forte charge instantanée du système, nous pouvons nous demander si elles ne sont pas corrélées avec une forte charge de plus longue durée. Pour cela, la figure 3.3 présente la charge et le pourcentage d'événements de surchauffe à travers les mois (Figure 3.3a), les jours (Figure 3.3b) et les heures (Figure 3.3c). La charge est calculée de la même manière que précédemment avec une unité de temps d'une minute. Puis, elle est agrégée par heure, par jour et par mois. Le pourcentage d'événements de surchauffe est calculé en prenant en compte le nombre d'événements de surchauffe arrivant pendant le mois, le jour ou l'heure considérés puis en le divisant par le nombre total d'événements de surchauffe sur la période considérée.

Alors que les événements de surchauffe ne semblent pas être corrélés avec une forte charge instantanée, ils semblent néanmoins corrélés avec une activité plus importante sur le système. En effet, le nombre d'événements de surchauffe est plus important lors

des heures de travail de 8h à 16h (Figure 3.3c) et durant les jours de travail du lundi au vendredi (Figure 3.3b). Il ne semble pas y avoir de corrélations entre la variation de charge entre les mois et les événements de surchauffe.

Nous pouvons conclure ici que les événements de surchauffe semblent être en rapport avec une activité plus importante sur le système. C'est-à-dire que plus l'utilisation du système est grande, plus la probabilité d'apparition des événements de surchauffe est importante. Pour analyser de manière plus précise cette utilisation du système et l'apparition des événements de surchauffe, nous allons dans le prochain paragraphe analyser de manière plus précise la corrélation entre les applications et les événements de surchauffe.

3.4.2 Influence des applications s'exécutant sur le système

Nous allons analyser ici les corrélations entre l'occurrence des événements de surchauffe, les applications et les caractéristiques de celles-ci.

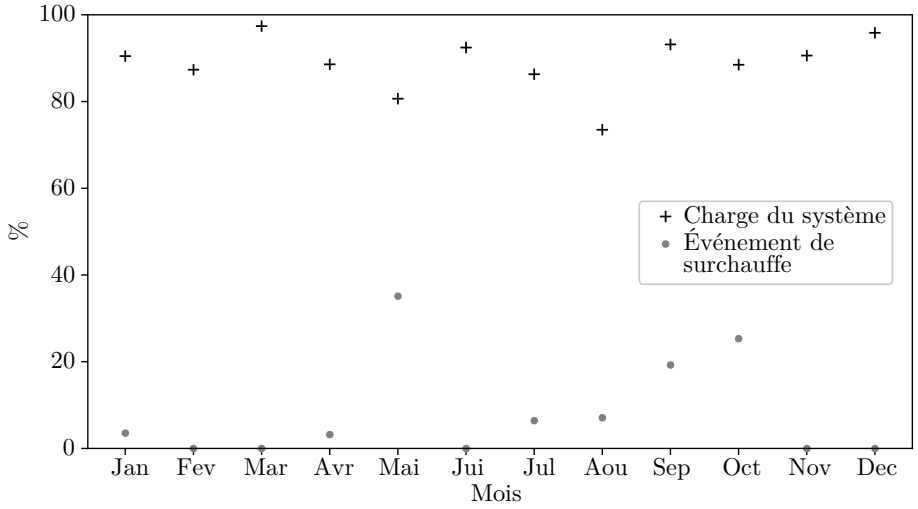
Pour réaliser cette étude, il est nécessaire de différencier une application et une instance de cette application. En effet, une même application peut être exécutée plusieurs fois avec des données différentes ou des paramètres différents (prévision météorologique sur différentes parties du monde par exemple). Chaque nouvelle exécution d'une application est appelé instance (*job*). Nous considérons par la suite que deux instances partageant le même nom correspondent à la même application. Pour déterminer le nom d'une instance, nous utilisons la base de données du gestionnaire de ressources. Le nom des instances est nettoyé afin de ne garder que les caractères alphabétiques. Par exemple nous considérons que les travaux "ocean_run_1" et "ocean_run_2" appartiennent à la même application "oceanrun".

Pour réaliser l'analyse de la corrélation entre les événements de surchauffe et les applications, la figure 3.4 présente 3 distributions à l'aide de cartes de chaleur (*heatmap*). Sur chaque figure, l'axe des abscisses représente la taille des instances (en considérant le nombre de ressources réservées) et l'axe des ordonnées représente la durée des instances en minute. Un type d'instance est défini par sa durée et sa taille. La figure 3.4a montre la distribution des instances soumises (comme un pourcentage du nombre total d'instances soumises) pour chaque type d'instances. La figure 3.4b présente le pourcentage de la charge du système générée par chaque type d'instance. Enfin, la figure 3.4c montre le pourcentage d'événements de surchauffe survenant pour chaque type d'instance.

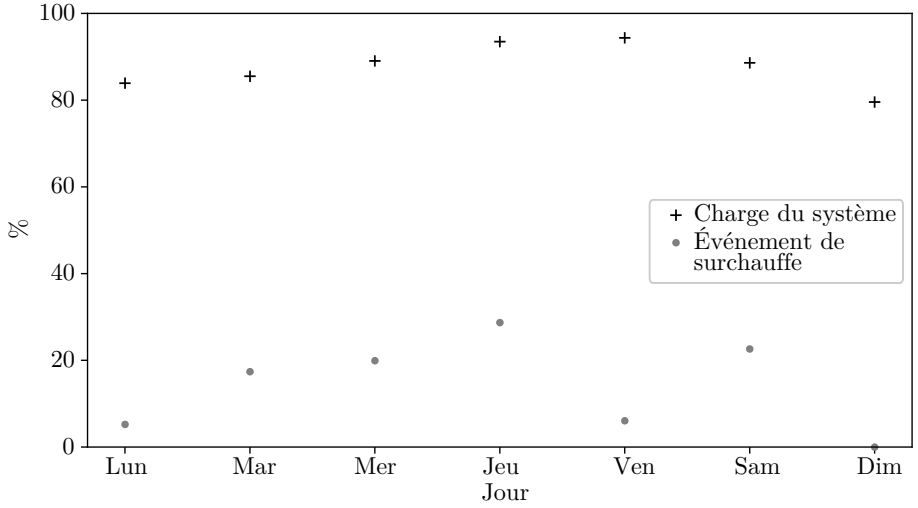
La première observation est que la plupart des instances soumises utilise un nœud et dure moins d'une minute (47% des travaux soumis) mais elles ne correspondent qu'à 0.16% de la charge du système. Il n'y a que très peu de *grosses* instances (moins de 1.41% des instances soumises a une durée d'exécution plus grande que 60 minutes et utilise plus de 16 nœuds) mais elles correspondent à la majorité de la charge totale (61.29%) du système. Les événements de surchauffe arrivent principalement (69%) dans le cas où les instances utilisent entre 8 et 32 nœuds et durent entre 20 et 60 minutes. Nous observons aussi qu'il n'y a pas d'événement de surchauffe survenant lorsqu'une instance utilise moins de 8 nœuds et dure moins 20 minutes. Cela signifie que dans notre cas d'étude, sous un seuil de ressources utilisées une surchauffe a une faible probabilité de survenir.

Pour mieux comprendre pourquoi au-dessus de ce seuil de ressources utilisées, le nombre d'événements de surchauffe n'est pas proportionnel à la charge induite, une analyse plus détaillée est requise.

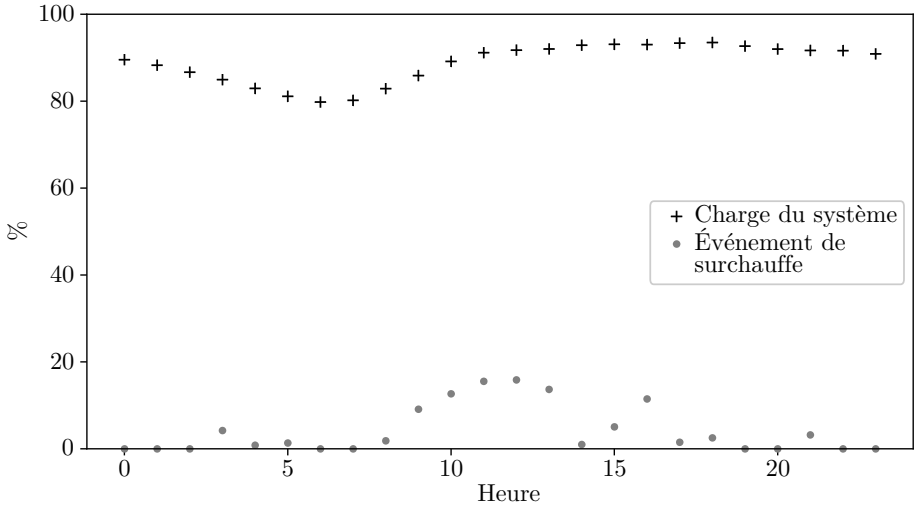
La figure 3.5 présente le pourcentage d'applications distinctes ayant subi les conséquences d'un événement de surchauffe en fonction du nombre de nœuds utilisés par



(a) Charge et pourcentage des événements de surchauffe selon le mois

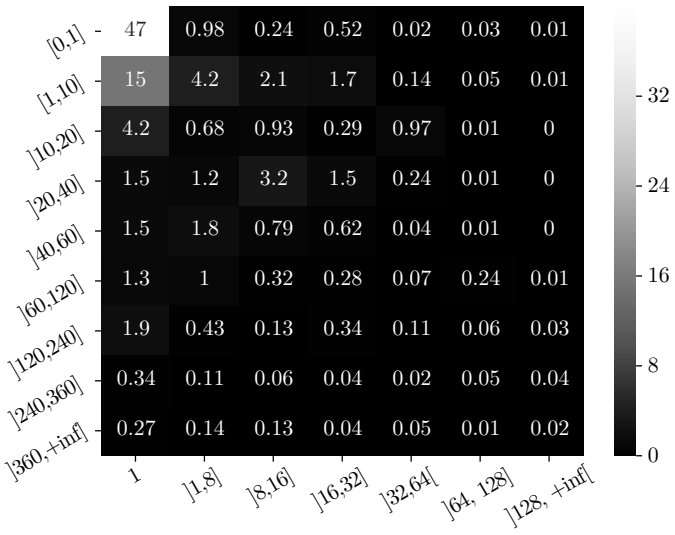


(b) Charge et pourcentage des événements de surchauffe selon le jour

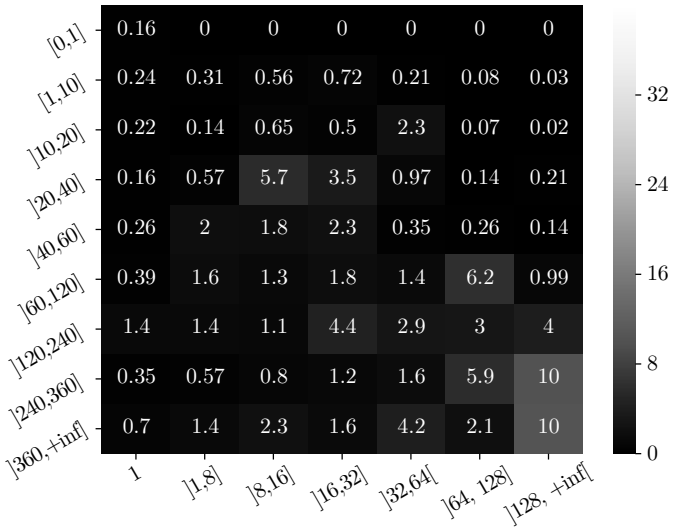


(c) Charge et pourcentage des événements de surchauffe selon l'heure de la journée

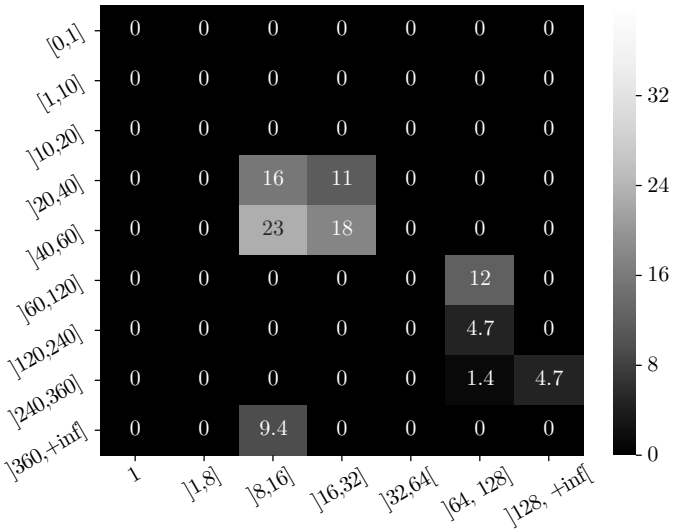
FIGURE 3.3 – Distribution de la charge et des événements de surchauffe.



(a) Instances soumises



(b) Charge



(c) Événements de surchauffe

FIGURE 3.4 – Distribution des instances soumises, de la charge et des événements de surchauffe en fonction du nombre de nœuds et du temps d'exécution des instances.

l'application. La figure 3.6 présente le même graphique mais en fonction du temps d'exécution de l'application. Ces figures considèrent uniquement les applications utilisant plus de 8 nœuds ou les applications durant plus de 20 minutes. Le pourcentage de chaque classe d'applications distinctes est calculé en utilisant le nombre d'applications distinctes pour une caractéristique donnée (temps d'exécution ou nombre de nœuds utilisés) divisé par le nombre total d'applications distinctes. Par exemple, si 52% des applications distinctes utilisent entre 8 et 16 nœuds cela signifie que sur un total de 4000 applications il y a 2080 applications distinctes qui utilisent entre 8 et 16 nœuds. En tout, il y a 18 applications ayant subi les conséquences d'un événement de surchauffe. La somme totale des pourcentages sur les figures peut être supérieure à 100% car une application peut utiliser un nombre de nœuds différents selon les instances démarrées. Cette application peut donc se retrouver dans plusieurs catégories.

La figure 3.5 et la figure 3.6 montrent une corrélation faible entre le nombre d'applications distinctes et le nombre d'applications distinctes ayant subi les conséquences d'un événement de surchauffe. Par exemple, selon la figure 3.5, les applications utilisant entre 8 et 16 nœuds représentent 52% des applications distinctes utilisant plus de 8 nœuds et 43% des applications distinctes affectées par un événement de surchauffe utilisent entre 8 et 16 nœuds.

Ces observations nous conduisent à penser que les événements de surchauffe sont plus probablement dus à des applications spécifiques et par conséquent plus il y a d'applications distinctes démarrées, plus la probabilité de démarrer une application capable de générer une surchauffe est grande. Cependant, nous pouvons voir 3 valeurs aberrantes dans les figures 3.5 et 3.6 : il n'y a pas d'application utilisant entre 32 et 64 nœuds affectée par un événement de surchauffe, et il n'y a que très peu d'applications fonctionnant entre 60 et 120 minutes et entre 120 et 240 minutes affectées par un événement de surchauffe. Les valeurs aberrantes s'expliquent car il y n'a que très peu d'applications affectées par un événement de surchauffe (18) et donc la probabilité est très faible de démarrer une application capable de générer une surchauffe.

Le tableau 3.1 fournit des détails à propos des applications : le nombre d'instances affectées par une surchauffe, le nombre d'instances total, le nombre d'événements de surchauffe arrivant durant une instance de l'application, la durée moyenne d'une instance de l'application et le nombre moyen de nœuds utilisés par une instance de l'application. Le tableau 3.1 montre que les applications utilisant un grand nombre de nœuds n'ont pas une probabilité plus élevée d'être affectées par un événement de surchauffe. Ce qui tend à confirmer que les événements de surchauffe ne sont pas des événements aléatoires. Cependant, pour la majorité des applications affectées par un événement de surchauffe, seul une petite partie des instances de ces applications est affectée (moins de 1% pour les instances de 11 des 18 applications). Cela nous indique qu'un ensemble de conditions doit être réuni pour générer une surchauffe.

En complément, nous avons aussi analysé la répartition physique des événements de surchauffe pour savoir s'il existait un point chaud dans le système ou des nœuds sur lesquels les événements se produisent plus régulièrement. Les 85 événements de surchauffe observés sont répartis sur 28 nœuds. Aucun de ces nœuds n'a généré un nombre élevé d'événements de surchauffe. Nous avons aussi comparé la charge de ces nœuds pour savoir s'ils étaient plus utilisés en moyenne que les autres nœuds du système. Nous n'avons trouvé aucune information permettant de conclure que les nœuds sur lesquels les surchauffes surviennent ont une charge plus élevée ou sont utilisés par des applications avec des caractéristiques spécifiques (applications avec une longue durée d'exécution par exemple).

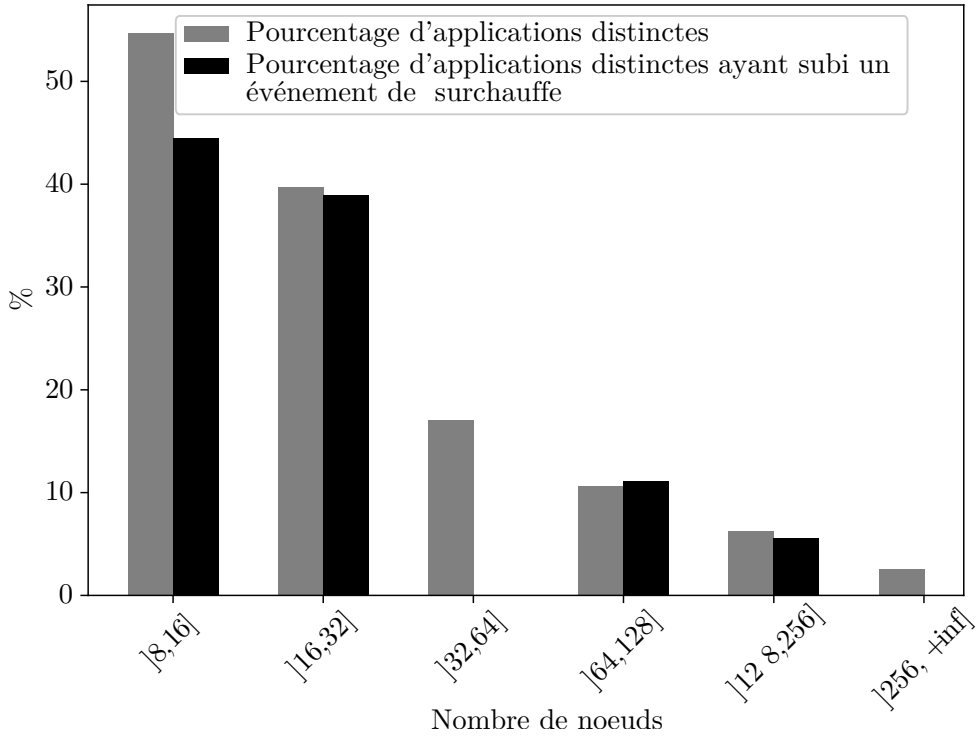


FIGURE 3.5 – Distribution des applications distinctes et distribution des applications distinctes ayant subi un événement de surchauffe en fonction du nombre de nœuds utilisés.

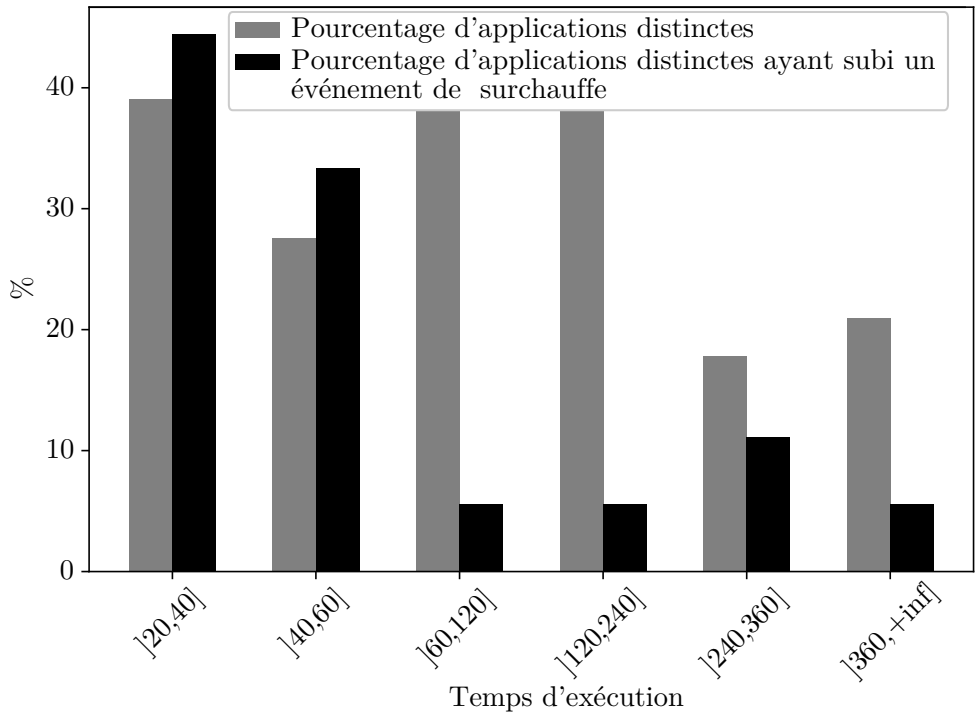


FIGURE 3.6 – Distribution des applications distinctes et distribution des applications distinctes ayant subi un événement de surchauffe en fonction du temps d'exécution.

Identifiant	Nombre d'instances total	Nombre d'instances ayant subi une surchauffe	Nombre d'événements de surchauffe	Durée moyenne d'une instance (minute)	Nombre moyen de nœuds utilisés
1	8	1 (12.5%)	1	34	16
2	114	1 (0.8%)	1	29	20
3	180	1 (0.5%)	7	29	16
4	200	1 (0.5%)	11	45	14
5	230	1 (0.4%)	21	69	12
6	198	1 (0.5%)	1	52	10
7	20	1 (5.0%)	14	453	12
8	954	1 (0.1%)	11	49	20
9	37	2 (5.4%)	18	62	97
10	158	1 (0.6%)	5	32	16
11	398	2 (0.5%)	10	40	20
12	14	3 (21.4%)	9	245	108
13	13	1 (7.6%)	7	47	16
14	508	2 (0.3%)	3	48	20
15	74	3 (4.0%)	7	316	241
16	36	1 (2.7%)	1	30	20
17	500	3 (0.6%)	12	48	20
18	664	1 (0.1%)	5	30	20

Tableau 3.1 – Applications ayant subi un événement de surchauffe.

Par conséquent, nous pouvons conclure qu'il est complexe de déterminer la cause racine exacte de ces événements à partir des données donc nous disposons. Une analyse plus poussée des applications incluant une analyse de leur code source et un profilage de leur comportement serait requise pour être capable d'identifier de manière plus précise les causes exactes d'apparition d'un événement de surchauffe.

3.4.3 Analyse de la température des processeurs

Enfin, la dernière corrélation que nous avons étudiée est la corrélation entre les événements de surchauffe et la température du processeur. Il paraît évident que les surchauffes sont liées à une augmentation de la température.

La température des processeurs est surveillée grâce à un ou plusieurs capteurs de température. A partir de ces capteurs, le processeur n'expose qu'une seule métrique : la *DTS* (*Digital Thermal Sensor*). Cette métrique indique la distance à la température maximale de fonctionnement du processeur. Elle est toujours négative ou égale à 0. Elle est exposée à l'utilisateur avec une précision d'un degré. Lorsqu'elle est égale à zéro, le processeur entre dans un état de surchauffe. Le calcul exact de cette métrique par le processeur dépend du modèle de celui-ci. Dans la suite, nous nous placerons d'un point de vue utilisateur de cette métrique. C'est-à-dire que nous ne chercherons pas à comprendre comment elle est calculée à partir des différents capteurs. Nous considérons simplement que la métrique est représentative de l'état du processeur et qu'elle est exacte. Dans notre cas cette métrique est collectée avec un taux d'échantillonnage d'une minute.

Il est donc intéressant d'étudier le comportement de la DTS avant une surchauffe du processeur et de voir l'influence du taux d'échantillonnage et de la précision de la température sur le comportement observé.

La figure 3.7 montre la variation de la température des processeurs avant les événements de surchauffe. Les événements de surchauffe surviennent au temps 0 sur l'axe des abscisses. La figure montre la température moyenne avant les événements de surchauffe et les percentiles (20% et 80%) associés. Les percentiles sont calculés indépendamment chaque minute.

Plusieurs observations peuvent être faites à l'aide de cette figure. La première est que la température moyenne des processeurs augmente plusieurs minutes avant l'événement de surchauffe. Cette augmentation est très visible en observant le 20% percentile. En effet, nous pouvons voir une augmentation nette de la température à partir de 40 minutes avant la surchauffe. La deuxième observation est que l'intervalle de variation de la température avant un événement de surchauffe est grand. Alors que 20% des processeurs ont une température inférieure à -20 40 minutes avant la surchauffe, 20% des processeurs ont une température comprise entre -5 et -3 degrés durant la période de 200 minutes que montre la figure avant la surchauffe. Enfin, la dernière observation est qu'au moment de la surchauffe l'ensemble des températures converge entre -3 et -4 degrés. Nous rappelons ici que le processeur entre dans un état de surchauffe quand la DTS (distance à la température maximale de fonctionnement) est égale à 0. Cela illustre que le taux d'échantillonnage choisi pour enregistrer la température et que la précision de la DTS (1 degré) ne sont pas suffisant pour décrire précisément la variation de la température.

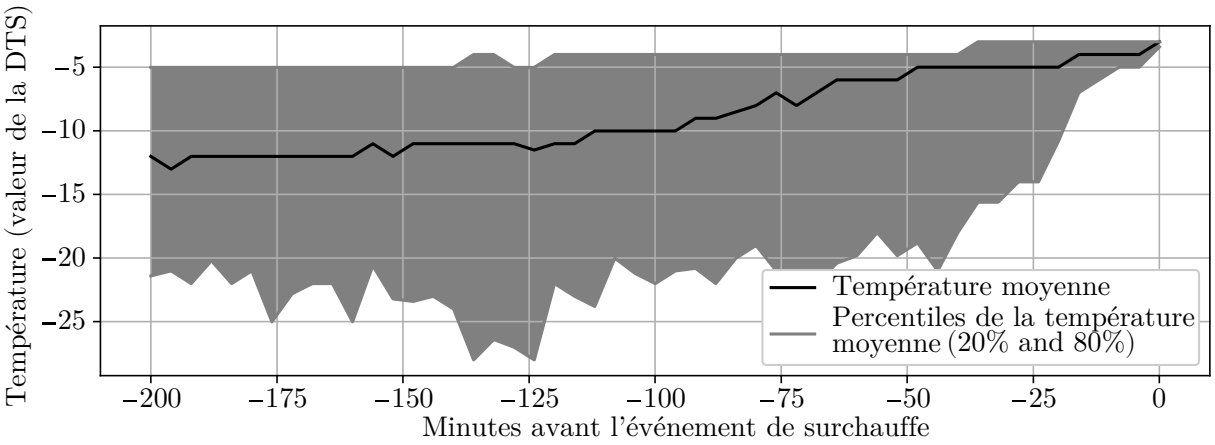


FIGURE 3.7 – Variation de la température du processeur avant les événements de surchauffe.

3.4.4 Synthèse

Nous pouvons conclure que les causes de la surchauffe sont complexes à mettre en évidence. Il apparaît que des applications spécifiques semblent être à l'origine des surchauffes. Cependant, ces applications ne génèrent pas des surchauffes à chaque nouveau démarrage. Ainsi, les surchauffes semblent être causées par plusieurs événements concomitants comme des instructions processeurs spécifiques utilisées par certaines applications et une charge globale forte du système. Pour finir, la seule métrique identifiée permettant de détecter des événements précurseurs dans l'apparition des événements de surchauffe est la température des processeurs. Dans la majorité des cas, celle-ci augmente de manière significative environ 40 minutes avant le début des événements de surchauffes.

3.5 Conséquences des surchauffes dans les systèmes HPC

Nous allons maintenant nous intéresser aux conséquences des surchauffes dans les systèmes HPC. Dans cette analyse, nous déterminerons les conséquences sur la disponibilité d'un système HPC en nous focalisant sur l'influence des surchauffes sur les applications distribuées utilisées dans les systèmes HPC. Nous commençons par expliquer les expérimentations menées, puis nous décrivons le protocole et les paramètres de ces expérimentations. Enfin, nous exposons les résultats obtenus.

3.5.1 Conditions d'expérimentation

La conséquence directe d'une surchauffe est une baisse de la fréquence du processeur. Cette baisse de fréquence entraîne une baisse de la capacité de calcul du processeur. Par la suite, un événement de surchauffe sera donc modélisé par une baisse de fréquence du processeur.

Il n'est cependant pas immédiat de déterminer l'influence de la baisse de la fréquence du processeur sur le temps d'exécution de l'application s'exécutant sur celui-ci. Nous devons donc déterminer l'influence de cette baisse en prenant en compte les spécificités des applications HPC.

La première spécificité est que le fonctionnement d'une application HPC n'est pas linéaire et alterne entre plusieurs phases. Par exemple, une application charge des données au démarrage depuis le système de fichiers, puis effectue une phase de calcul et une

phase d'échange des données entre les processeurs utilisés et ensuite écrit les résultats sur le système de fichiers. Nous pouvons donc nous attendre à des influences différentes d'une baisse de fréquence du processeur selon la phase de l'application. En effet, lors de ces différentes phases, le système limitant ne sera pas toujours le processeur. Lors de l'accès au système de fichiers, le système limitant sera le système de fichiers alors que le réseau sera le système limitant durant l'échange des données entre les processeurs utilisés par l'application. Nous pouvons donc penser qu'une baisse de fréquence du processeur lorsqu'une écriture sur le système de fichiers n'aura que peu d'influence (le processeur n'étant pas l'élément limitant) alors qu'une baisse de fréquence durant une phase de calcul aura une influence significative (le processeur étant cette fois ci l'élément limitant). Pour la suite de nos expérimentations, nous partons de l'hypothèse qu'une surchauffe ne peut pas survenir lors qu'une longue phase d'accès au système de fichiers car le processeur est quasiment dans un état de repos et ne produit donc que très peu de chaleur. Par conséquent, nous allons évaluer quelle est l'influence d'une baisse de fréquence lors du fonctionnement d'une application en dehors des phases d'accès au système de fichiers.

La deuxième spécificité est la distribution des applications HPC. En effet, une application HPC utilise plusieurs nœuds en parallèle pour calculer les résultats attendus. Il est donc nécessaire d'estimer l'influence d'une baisse de fréquence d'un processeur utilisé par l'application dans le cas où celle-ci en utilise plusieurs.

3.5.2 Protocole et paramètres d'expériences

Pour évaluer l'influence d'une baisse de fréquence du processeur sur des applications HPC, nous utilisons un ensemble de 6 applications représentatives des applications utilisées dans un système HPC. Ces applications proviennent de la suite de tests de performance *Trinity* fournit par le NERSC [5]. Ces applications sont les suivantes : AMG, coMD, GTC, MILC, miniFE et miniGhost. Chaque application est représentative d'un type de calcul effectué par les applications utilisées dans un système HPC. AMG est une application proposant un solveur algébrique multi-grilles pour des maillages physiques non structurés, coMD est une application se focalisant sur la dynamique moléculaires, GTC calcule des fusions magnétiques en se basant sur la méthode de *Particle-in-cell*, MILC calcule de la chromodynamique quantique en treillis, miniFE effectue du calcul implicite d'éléments finis non structurés, et miniGhost calcule les cœfficients des schémas de différences finies explicites ou implicites. Chaque application est distribuée. Pour évaluer l'influence d'une baisse de fréquence d'un processeur sur l'ensemble des processeurs utilisés, chaque application est exécutée sur plusieurs nœuds. Lors de cette exécution, la fréquence d'un processeur sera réduite. Nous étudions ensuite la différence de temps d'exécution entre une exécution normale et une exécution avec une baisse de fréquence pour en déduire l'influence de celle-ci.

Les expériences utilisent un système composé de 120 nœuds. Chaque nœud est équipé d'un processeur Intel Xeon X3440 4 cœurs et de 16 Go de mémoire vive. Les nœuds sont connectés entre eux en utilisant un réseau InfiniBand 20 Gbps. Chaque nœud fonctionne sous Debian 9 avec un noyau en version 4.9. Le gouverneur de puissance du processeur est réglé sur "performance maximale". Nous utilisons MPICH v3.2.1 comme bibliothèque MPI. Nous utilisons le paquet *cpufreq* [6] pour fixer la fréquence du processeur. Sur le processeur utilisé, la fréquence peut être sélectionnée entre 2.53GHz et 1.2GHz avec différents paliers.

5. https ://www.nersc.gov/assets/Trinity–NERSC-8-RFP/Documents/N8BmkInstructJune13.pdf
6. https ://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

Nous testons plusieurs baisses de fréquence sur différentes durées. Nous avons sélectionné 4 fréquences différentes pour nos tests : 2.53GHz (la fréquence maximale), 2.27Ghz (90% de la fréquence maximale), 1.87GHz (74% de la fréquence maximale) et 1.2GHz (47% de la fréquence maximale, correspondant à la fréquence minimale du processeur). Les durées de la baisse de fréquence testées sont de 30, 90 et 150 secondes. Le protocole de test est le suivant : un processeur utilisé par l'application est sélectionné de manière aléatoire puis sa fréquence est réduite après 100 secondes d'exécution pour éviter la phase d'initialisation de l'application. Chaque test est répété 5 fois pour chaque configuration en sélectionnant à chaque fois un nouveau processeur aléatoirement.

Le tableau 3.2 présente les paramètres utilisés pour chaque application. Seuls les paramètres différents des paramètres par défaut utilisés par la suite de tests de performance *Trinity* sont spécifiés dans le tableau. Les paramètres des applications sont sélectionnés pour que le temps d'exécution des applications soit de l'ordre de quelques centaines de secondes. Nous utilisons différents nombres de rangs MPI pour estimer la variation de l'influence de la baisse de la fréquence processeur dans le cas d'une distribution plus ou moins importante. Le nombre de rangs est défini selon les configurations possibles de l'application. Un rang MPI correspond à un *processus* de l'application (un *processus* est ici utilisé pour désigner un *processus* au sens programme informatique). Un processus est assigné à un cœur d'un processeur. Nous remplissons au maximum chaque processeur. Ainsi, dans un test avec 128 rangs, nous utilisons 32 processeurs avec 4 rangs MPI par processeur. Pour chaque application, les paramètres sont adaptés pour garder une taille locale constante du problème pour chaque rang (*weak scaling*). Dans le cas où le nombre de rangs augmente de 128 à 256, chaque rang MPI garde la même taille de données pour son calcul, mais la taille du calcul global sera deux fois plus importante. Le tableau montre aussi les profils de chaque application pour 128 processus. Le pourcentage de communication MPI est le pourcentage de temps utilisé par l'application dans les communications (envoi et réception de données entre les différents nœuds), l'empreinte mémoire détaille la consommation de mémoire vive de chaque rang de l'application, le temps d'exécution montre le temps d'exécution typique de l'application en utilisant les paramètres fournis, enfin la dernière colonne fournit les paramètres différents des paramètres par défaut utilisés pour chaque application. L'empreinte mémoire, le pourcentage de communication MPI et le temps d'exécution sont fournis par la bibliothèque IPM [7].

La première remarque que nous pouvons faire est que chaque application a un profil différent : les communications varient entre 4.28% et 11.23% du temps total et l'empreinte mémoire varie entre 0.09 Go et 2.8 Go. Ainsi, nous pouvons nous attendre à des conséquences différentes de la baisse de fréquence d'un processeur selon le profil de l'application.

Pour compléter l'analyse de la différence de temps d'exécution entre une exécution normale et une exécution avec une baisse de fréquence, nous analysons le profil d'accès à la mémoire vive de chaque application. Nous souhaitons savoir en analysant ce profil si les accès à la mémoire vive peuvent se répercuter sur l'influence d'une baisse de fréquence.

Pour cette analyse, un autre système est utilisé. En effet, les processeurs Intel Xeon X3440 ne disposent pas des compteurs spécifiques nous permettant de tracer le profil d'accès à la mémoire vive. Nous utilisons un système équipé de deux processeurs Xeon E5-2630 v3 avec 8 cœurs et 128 Go de mémoire vive. Nous utilisons l'outil de surveillance Intel PCM [8] pour tracer le profil des accès à la mémoire vive avec un taux d'échantillonnage d'un point par seconde. De la même manière que précédemment, nous fixons la fréquence du processeur

7. https ://github.com/nerscadmin/IPM numéro de commit 02f0cdc
8. https ://software.intel.com/en-us/articles/intel-performance-counter-monitor

Application	Nombre de rangs MPI	Pourcentage de communication MPI (128 rangs)	Empreinte memoire (128 rangs)	Temps d'exécution (128 rangs)	Paramètres (128 rangs)
AMG	128-256-448	8.6%	0.52 Go/rang	347s	solveur : 1 / taille : (160,160,160)
coMD	128-256-480	4.28%	0.8 Go/rang	294s	taille : (250,250,200)
GTC	128-256-480	8.91%	2.8 Go/rang	605s	étapes : 30 / npartdom : 2 / micell : 200 / mecell : 200
MILC	128-256	11.23%	0.09 Go/rang	280s	taille : (64,32,32,48)
miniFE	128-256-480	6.56%	0.49 Go/rang	362s	étapes : 1600 / taille : (600,600,600)
miniGhost	128-256-448	9.02%	2.65 Go/rang	453s	étapes : 65 / taille : (200,200,200)

Tableau 3.2 – Configuration et profil des applications testées.

Application	2.27 GHz (90% de la fréquence maximale)	1.87 GHz (74% de la fréquence maximale)	1.2 GHz (47% de la fréquence maximale)
AMG	12%	24%	49%
coMD	11%	26%	53%
GTC	12%	27%	55%
MILC	0%	8%	19%
miniFE	0%	0%	0%
miniGhost	0%	0%	25%

Tableau 3.3 – Influence de la baisse de la fréquence d'un processeur sur le temps d'exécution d'une application distribuée.

à 3 fréquences différentes : 2.4GHz (fréquence maximale), 1.9GHz (79% de la fréquence maximale) et 1.3GHz (54% de la fréquence maximale, correspondant à la fréquence minimale du processeur). Pour chaque application, la baisse de fréquence est appliquée 4 minutes après le démarrage de celle-ci. L'application est démarrée en utilisant 16 rangs MPI (un par cœur), et nous mettons à jour les paramètres des applications pour avoir un temps d'exécution supérieur à 10 minutes. Il est à noter que nous distribuons volontairement l'application en n'utilisant pas le réseau d'interconnexion (les deux processeurs sont situés sur la même carte mère). En effet, nous souhaitons limiter l'influence de celui-ci dans l'analyse des répercussions des accès à la mémoire vive sur les conséquences d'une baisse de fréquence.

3.5.3 Influence de la baisse de fréquence sur le temps d'exécution de l'application

Les résultats de nos expérimentations concernant la différence de temps d'exécution entre une exécution normale et une exécution avec une baisse de fréquence sont donnés dans le tableau 3.3. Nous présentons les résultats en agrégeant les résultats des différents nombres de rangs MPI de chaque application. En effet, nos résultats ont montré que l'influence est identique quel que soit le nombre de rangs utilisé. Nous présentons les résultats comme l'augmentation du temps d'exécution de l'application par rapport à la durée de la baisse de fréquence. Par exemple, l'augmentation du temps d'exécution de 27% pour l'application GTC signifie que pour une baisse de fréquence d'une durée de 100 secondes, le temps d'exécution de l'application est augmenté de 27 secondes.

Nous observons 3 comportements différents selon nos applications : i) miniFE n'est pas sensible à la baisse de fréquence ; ii) miniGhost et MILC sont sensibles uniquement à une baisse de fréquence importante ; iii) coMD, GTC, et AMG sont très sensibles à la baisse de fréquence. Dans le cas de ces 3 dernières applications, la baisse de performance est directement proportionnelle à la baisse de fréquence survenant sur un nœud. Il est à noter que nous n'avons pas trouvé de corrélations entre le pourcentage de communication ou l'empreinte mémoire et l'influence de la baisse de fréquence.

Ces résultats montrent que pour certaines applications, une surchauffe (simulée par la baisse de fréquence ici) affectant un processeur peut avoir une influence importante sur le temps d'exécution de l'application indépendamment de sa distribution sur un grand nombre de nœuds.

Application	1.9GHz (79% de la fréquence maximale)	1.3GHz (54% de la fréquence maximale)
AMG	21%	43%
coMD	20%	50%
GTC	24%	50%
MILC	6%	25%
miniFE	0%	6%
miniGhost	15%	36%

Tableau 3.4 – Influence de la baisse de la fréquence d'un processeur sur le temps d'exécution d'une application utilisant un seul nœud.

3.5.4 Analyse des profils des applications

Pour mieux comprendre pourquoi l'influence de la baisse de fréquence est différente selon les applications, nous avons analysé le profil des accès mémoire de ces applications. L'analyse de ce profil nous permet de déterminer les corrélations entre les accès mémoires et l'influence la baisse de fréquence selon les applications.

La figure 3.8 présente les profils des accès mémoires des différentes applications et le tableau 3.4 présente l'augmentation du temps d'exécution des applications par rapport à la durée de la baisse de fréquence. La bande passante maximale disponible d'après la documentation d'Intel [9] est de 42.6 Go/s par processeur et donc de 85.2 Go/s pour notre nœud composé de deux processeurs. Nous pouvons voir sur la figure 3.8 que l'application miniFE atteint le maximum de bande passante disponible sur notre système et par conséquent l'élément limitant se trouve être la bande passante mémoire pour cette application et non le processeur. Quand la fréquence du processeur baisse fortement, c'est celui-ci qui devient l'élément limitant et la bande passante maximale de la mémoire n'est plus atteinte. Le tableau 3.4 montre que pour l'application miniFE, seul une importante baisse de fréquence peut induire une augmentation du temps d'exécution de l'application. Ceci montre que si la bande passante mémoire est l'élément limitant, une baisse de fréquence du processeur n'aura pas d'influence sur le temps d'exécution de l'application. Pour les 5 autres applications, la bande passante maximale n'étant pas atteinte, elles sont donc limitées par le processeur. Cependant, l'importance de l'influence varie. Pour AMG, coMD et GTC, l'influence est proportionnelle à baisse de fréquence alors que pour MILC et miniGhost l'influence est plus faible. Pour ces deux dernières, il est possible d'un autre sous-système limite leur performance.

3.6 Synthèse

À travers cette étude nous avons analysé les données de supervision disponibles sur le supercalculateur de production de DKRZ pour déterminer les causes des événements de surchauffe dans les systèmes HPC.

Nous avons montré que les événements de surchauffe ne sont pas des événements aléatoires survenant dans un système HPC.

9. https ://ark.intel.com/fr/products/64593/Intel-Xeon-Processor-E5-263 0-15M-Cache-2 30-GHz-7 20-GTs-Intel-QPI

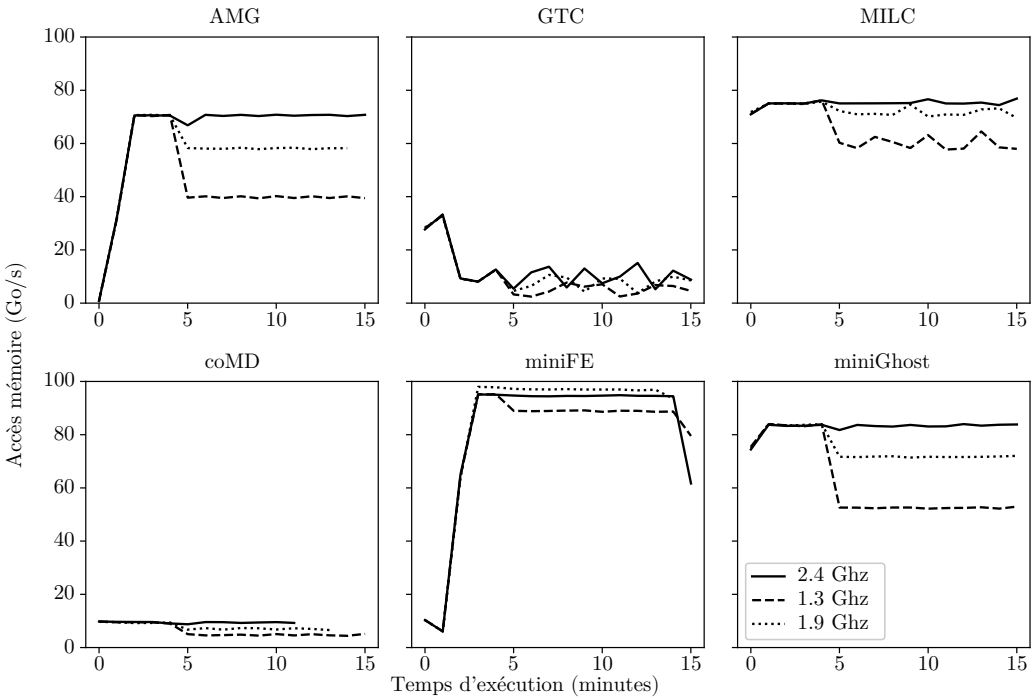


FIGURE 3.8 – Profils d'accès mémoire des applications.

L'analyse des corrélations avec les caractéristiques du système nous permet de conclure qu'ils ne sont pas dus à une forte charge ou à un point chaud du système.

L'analyse des corrélations avec les applications fonctionnant sur le système montre qu'ils ne semblent pas liés à des caractéristiques d'applications spécifiques (applications fortement distribuées avec un temps d'exécution long par exemple). Cependant, cette analyse permet de mettre en évidence que plus il y a d'applications distinctes démarrées, plus la probabilité de démarrer une application capable de générer une surchauffe est grande. Les événements de surchauffe semblent donc être dus à des applications spécifiques démarrant dans des conditions particulières. L'analyse plus précise de ces applications n'est pas réalisable dans notre cas car nous ne disposons pas d'information à propos de celles-ci dans les données disponibles que nous analysons.

L'analyse de l'évolution de la température du processeur avant un événement de surchauffe montre que la température peut être utilisée comme un indicateur d'un événement de surchauffe. En effet, la majorité des températures commence à augmenter à partir de 40 minutes avant une surchauffe et l'ensemble des températures des processeurs observées converge entre -5 et -3 degrés (en relatif par rapport à la température maximale des processeurs) lors de l'événement de surchauffe. En conclusion, la température semble être donc le meilleur indicateur pour détecter les événements précurseurs d'une surchauffe.

La deuxième partie de cette étude est consacrée à l'étude de l'influence des surchauffes sur le temps d'exécution des applications HPC. Nos expérimentations menées sur des applications représentatives des applications utilisées dans les systèmes HPC montrent l'influence négative des événements de surchauffe sur le temps d'exécution. Dans le cas d'une application limitée par le processeur, l'augmentation du temps d'exécution de l'application est directement proportionnelle à la réduction de la fréquence du processeur induite par la surchauffe, indépendamment de la distribution de l'application. Nos expérimentations montrent aussi que l'influence peut être différente selon les applications et leurs caractéristiques. L'évaluation des répercussions des accès mémoires sur l'influence de la baisse en

fréquence montre que les surchauffes influencent moins le temps des applications limitées par les accès mémoires.

En conclusion, les surchauffes sont des événements survenant sur les systèmes HPC qui peuvent influer grandement sur le temps d'exécution des applications HPC. Ces événements sont probablement dus à des applications spécifiques démarrant dans des conditions particulières. La température du processeur est la meilleure métrique disponible dans nos données de supervision pour être utilisée comme un indicateur d'un événement de surchauffe.

Chapitre 4

Solution pour la prédiction des surchauffes des processeurs dans un système HPC

Comme nous l'avons montré dans le chapitre 3, les surchauffes des processeurs dans les systèmes HPC ont des conséquences sur le temps d'exécution des applications et peuvent par conséquent réduire la disponibilité de ces systèmes. Il est donc important de pouvoir trouver une solution permettant de mitiger les conséquences des surchauffes dans les systèmes HPC.

Dans ce chapitre, nous présentons dans un premier la solution prédictive développée permettant de prédire les événements de surchauffe à l'aide des analyses réalisées et des conclusions faites dans le chapitre 3. Puis dans un second temps, nous présentons les actions envisagées permettant d'éviter les surchauffes en utilisant les prédictions réalisées par la solution. La solution prédictive proposée s'appuie sur l'analyse des données de température des processeurs en utilisant des méthodes d'apprentissage machine pour prédire les prochains événements de surchauffe.

Nous commençons par détailler les problématiques liées à la construction d'une solution prédictive et l'apport de l'état de l'art dans la construction de la solution dans la section 4.1. Puis, dans la section 4.2, nous expliquons la solution développée et son fonctionnement en décrivant les principales étapes nécessaires à la mise en place de celle-ci. Nous exposons ensuite les résultats de cette solution sans tenir compte d'une possible action préventive dans la section 4.3. Puis, nous décrivons les actions préventives envisagées et le calcul du coût de la surchauffe et nous justifions leur choix par des résultats de réduction du coût de la surchauffe dans la section 4.4. Nous discutons de nos résultats dans la section 4.5 et nous dressons la synthèse de ces travaux dans la section 4.6.

4.1 Problématique

La construction d'une solution de prédiction des surchauffes impose de multiples contraintes. En effet, notre solution s'insère dans un système HPC qui est un système complexe. Elle doit donc prendre en compte cette complexité dans son développement pour parvenir à obtenir des résultats fiables. En complément de cette solution, des actions doivent être entreprises pour éviter ou diminuer les conséquences de la surchauffe une fois l'événement de surchauffe correctement prédit. De la même façon que pour la solution de prédiction, les actions envisagées doivent prendre en compte la complexité d'un système

HPC et des applications fonctionnant sur ce système pour pouvoir être efficaces. Nous commençons cette section par introduire l'ensemble des défis liés à la construction d'une solution prédictive, puis nous détaillons les travaux présentés dans l'état de l'art pouvant aider à l'élaboration de notre solution.

4.1.1 Les défis

Pour développer une solution permettant de prédire efficacement les surchauffes, il est nécessaire de trouver des solutions à différents défis. Nous décrivons ici les principaux défis liés à l'élaboration de cette solution.

Prédire les surchauffes suffisamment en avance

Le premier défi à relever est lié à l'utilisation de la solution pour réduire le coût des surchauffes. En effet, la prédiction seule des surchauffes n'est pas suffisante. Il est nécessaire que ces prévisions soient réalisées suffisamment en avance pour que des actions puissent être prises pour mitiger le coût des surchauffes sur le système HPC. Il est donc important de considérer à la fois le temps nécessaire à l'action pour être efficace mais également la durée d'efficacité de l'action. Par conséquent, le développement de la solution doit tenir compte du temps d'avance nécessaire pour que les actions puissent mitiger le coût des surchauffes.

Garantir des résultats satisfaisants tout en limitant les faux positifs

Il est évident que la considération du temps d'avance nécessaire des prédictions par rapport aux actions doit se faire en garantissant des résultats satisfaisants. Ces résultats doivent permettre à la solution d'être capable de réduire le coût des surchauffes grâce aux actions qui seront entreprises. La qualité des résultats doit s'évaluer sur deux critères. Le premier est la capacité de la solution à prédire correctement l'ensemble des surchauffes (caractérisée par la métrique de rappel). Le deuxième est la capacité de la solution à limiter les faux positifs (caractérisée par la métrique de précision), c'est-à-dire limiter le nombre de fausses prédictions (*i.e.*, qui n'ont pas de liens avec un événement de surchauffe).

Exploiter les données disponibles

La construction et l'évaluation de la solution utilisent des données provenant d'un système HPC actuellement utilisé en production (section 3.1). L'utilisation de ces données implique que nous devons utiliser des données collectées en tenant compte de certaines contraintes pratiques. Il est, en effet, compliqué de collecter les données de température à la fréquence d'évaluation du processeur de celle-ci (environ toutes les millisecondes) sur l'ensemble des nœuds du système par exemple.

Nous avons montré dans le chapitre 3 que la seule donnée collectée pouvant décrire les surchauffes est la température du processeur. Dans notre cas, le choix des administrateurs du système HPC a été de collecter la température des processeurs avec une fréquence d'échantillonnage d'une minute. Ce faible taux d'échantillonnage et la précision de la température fournie par le processeur (1°C) nous empêchent de pouvoir directement détecter les surchauffes sur les données de température (paragraphe 3.4.3). Il sera donc nécessaire de tenir compte de ces caractéristiques lors de la construction de la méthode pour pouvoir fournir des résultats fiables malgré des données peu précises.

De plus, pour pouvoir utiliser des méthodes d'apprentissage machine pour la prédiction des surchauffes, il est nécessaire de prendre en compte aussi le nombre des événements de surchauffe. En effet, dans le système HPC étudié, il y a peu d'exemples de surchauffe par an (moins de 100 dans notre cas). Ce faible nombre d'exemples nous conduit à utiliser des méthodes d'apprentissage machine adaptées à des apprentissages avec une faible quantité d'exemples et exclut par exemple l'utilisation des réseaux de neurones (paragraphe 2.3.3). Pour finir, les données de température sont des séries temporelles, nous devons donc utiliser des méthodes adaptées à l'analyse de ce type de données.

Transparence pour les applications

Pour faciliter le déploiement et l'utilisation de notre solution sur des systèmes HPC, nous pensons qu'il est nécessaire d'être transparent d'un point de vue utilisateur. C'est-à-dire que nous ne devons pas imposer à l'utilisateur de notre solution d'intégrer des modifications dans son application pour que notre solution puisse fonctionner. Nous devons donc être au maximum agnostiques au système lors de la construction de notre solution.

Surcoût de la solution pour le système HPC

Pour que notre solution et les actions entreprises soient capables de réduire le coût des surchauffes, le coût de la solution pour le système doit être minimisé. Pour cela, nous devons prendre le coût direct de calcul de la solution mais également son placement dans le système.

Le placement de la solution dans le nœud de calcul nous permet de proposer une solution sans aucun pré-requis d'installation (communication sur le réseau, déploiement de nœuds dédiés à cette tâche, etc.) et une mise en place simple. De plus, cela permet à la solution de passer simplement à l'échelle : chaque nœud de calcul réalise les prédictions localement en utilisant la température de son ou ses processeurs. Cependant, cela impose une méthode très légère en calcul car elle ne devra pas perturber les applications s'exécutant sur celui-ci.

En résumé, nous devons donc développer une solution légère en calcul permettant d'analyser des séries temporelles avec un faible taux d'échantillonnage en ayant peu de données d'exemples. Nous présentons maintenant les travaux de l'état de l'art utilisés pour construire notre solution.

4.1.2 Etat de l'art

Les données de température analysées représentent des séries temporelles. Nous allons donc nous appuyer sur l'état de l'art de l'analyse des séries temporelles pour la construction de notre solution. Nous commencerons par expliquer les deux approches possibles identifiées pour la prédiction des surchauffes, puis nous détaillerons les travaux de l'état de l'art sur l'analyse des séries temporelles et sur la prédiction des surchauffes des processeurs, et leurs apports dans nos travaux.

Approches pour la prédiction de surchauffes

Deux types d'approches peuvent être envisagés pour prédire les surchauffes. La première est de prédire les futures valeurs de la température du processeur. Si les prédictions des futures valeurs de la température du processeur dépassent un seuil, nous serons capables

de prédire qu'une surchauffe va se produire sur le processeur. La deuxième approche est d'analyser les variations de la température avant une surchauffe afin de détecter des signes précurseurs dans les variations de la température (une augmentation particulière de celle-ci par exemple).

Les deux approches sont évaluées dans ce chapitre. La solution construite utilise l'analyse des variations de la température pour prédire une surchauffe. Cette analyse tire parti des conclusions des travaux présentés dans l'état de l'art sur l'analyse des séries temporelles que nous détaillons dans le prochain paragraphe.

L'évaluation de l'approche utilisant une prédiction de la température et un seuil est réalisée en utilisant un oracle. Cela nous permet d'évaluer simplement une approche permettant une prédiction parfaite des prochaines valeurs de la température. La construction d'une solution réalisant la prédiction des températures est un travail à part entière. Cependant, l'utilisation d'une méthode présentant une prédiction parfaite (l'oracle) nous permet de fournir une borne supérieure aux résultats de ce type de solution (section 4.3.2).

Analyse des séries temporelles

Il existe de nombreux travaux traitant de la classification des séries temporelles [6, 1, 26, 99]. Comme le montre le travail d'Aghabozorgi et al. [1], la classification des séries temporelles est découpée en deux composants principaux en plus de l'algorithme de classification lui-même : un composant de réduction de dimension et un composant de mesure de similarité (aussi appelé extraction des caractéristiques). Dans notre cas, la réduction de dimension est simplement réalisée en utilisant des fenêtres glissantes. Par conséquent, nous ne discutons ici que des deux points restants : l'algorithme de classification et la mesure de similarité.

Des études récentes [6, 26] montrent que beaucoup d'algorithmes ont été proposés pour quantifier la similarité entre deux séries temporelles. Toutefois, ces études montrent que l'algorithme de Dynamic Time Warping (DTW) fournit un compromis acceptable entre la précision de la mesure et le coût de calcul. En utilisant ce constat, nous avons choisi l'algorithme de DTW pour notre solution. Cette méthode utilise ici la distance entre la série temporelle étudiée et les autres séries pour caractériser la série temporelle étudiée.

Pour comprendre le fonctionnement de l'algorithme de DTW, nous allons le décrire brièvement et le comparer à une autre méthode permettant l'extraction des caractéristiques en utilisant la distance, la distance euclidienne. La distance euclidienne calcule la distance entre deux points qui correspondent au même instant t, puis la somme des distances sur l'ensemble de la série est réalisée. Le principal inconvénient de cette méthode est qu'elle est sensible aux déformations temporelles. Si les deux séries sont décalées dans le temps (exemple, un sinus et cosinus), la distance euclidienne sera grande or les deux séries se *ressemblent* (elles sont même identiques, décalées de $\frac{\pi}{2}$). La DTW peut être utilisée pour mieux caractériser les séries temporelles. En effet, elle ne calcule pas la distance entre deux points au même instant t mais minimise la distance entre un instant t de la première série et un intervalle d'instants $[t - x, t + x]$ de la deuxième série.

Comparons le calcul de la DTW et la distance euclidienne sur un exemple. Prenons les deux séries temporelles suivantes : $t_1 = [1, 2, 3, 4, 5]$ et $t_2 = [0, 1, 2, 3, 4]$. La première série est décalée d'une unité par rapport à la seconde.

i=4	4	3	2	1	**0**	**1**
i=3	3	2	1	**0**	1	2
i=2	2	1	**0**	1	2	3
i=1	1	**0**	1	2	3	4
i=0	0	**1**	2	3	4	5
index	t_1 / t_2	1	2	3	4	5
	index	j=0	j=1	j=2	j=3	j=4

Tableau 4.1 – Grille de calcul utilisée par la DTW.

La distance euclidienne est définie par :

$$E_d(t_1, t_2) = \sqrt{\sum_{i=1}^{n}(t_{1i} - t_{2i})^2}$$

En utilisant notre exemple, nous avons :

$$E_d(t_1, t_2) = \sqrt{(1-2)^2 + (2-1)^2 + (3-2)^2 + (4-3)^2 + (5-4)^2} = 2.23$$

Calculons maintenant la DTW. Le calcul fait appel à une grille. Cette grille est présentée par le tableau 4.1. La colonne de gauche est la série temporelle t_1, celle de droite est la série temporelle t_2. Chaque cellule de la grille est la distance entre le point i de t_1 et le point j de t_2. Cette distance est définie par :

$$d(t_1[i]; t_2[j]) = |t_1[i] - t_2[j]|$$

La DTW cherche le chemin minimum à travers cette grille en partant du point $(0, 0)$ jusqu'au point (n, m) avec n la longueur de t_1 et m la longueur de t_2. Le chemin minimum (représenté par les valeurs de distance en gras dans le tableau 4.1) est donné par :

$$p_{min} = argmin(D(t_1, t_2))$$

avec

$$D(t_1, t_2) = \frac{\sum_{i=0,j=0}^{i=n,j=m} d(t_1[i]; t_2[j]) \times w_{ij}}{\sum_{i=0,j=0}^{i=n,j=m} w_{ij}} \text{ la distance}$$

w_{ij} le coefficient de poids

En fixant le cœfficient de poids à $w_{ij} = 1$, la DTW vaut

$$DTW(t_1, t_2) = \frac{1+0+0+0+0+1}{5} = 0.4$$

Ainsi, la distance calculée avec la DTW permet de mieux rendre compte de la ressemblance des séries par rapport à la distance calculée avec la distance euclidienne.

Le choix de l'algorithme de classification est plus spécifique au problème traité [3]. Ainsi, il n'existe pas de consensus sur le *meilleur* algorithme à utiliser dans tous les cas. Des travaux dressent cependant des directions. Xing et al. [99] pointent l'algorithme des k

plus proches voisins (kNN) [20] comme un point de départ. Nous incluons en complément de cet algorithme d'autres algorithmes de l'état de l'art : arbre décisionnel [82], forêt aléatoire [12] et Gradient Boosting Tree [35]. Notre but sera donc d'évaluer lequel de ces algorithmes permet d'obtenir les meilleurs résultats dans notre cas d'étude.

En conclusion, l'état de l'art nous fournit les bases pour la construction de notre solution permettant de prédire les surchauffes en utilisant l'analyse de la température des processeurs. Il est donc nécessaire d'effectuer d'abord une extraction des caractéristiques de la série puis une classification de ces caractéristiques à l'aide d'un algorithme d'apprentissage machine.

Prédictions des surchauffes et défaillances dans les systèmes HPC

Nous allons maintenant présenter les travaux qui s'articulent autour de la prédiction des événements de surchauffe ou des défaillances dans les systèmes HPC. Nous choisissons de nous focaliser sur les travaux utilisant des données temporelles ayant un lien avec la température pour la prédiction des défaillances ou des surchauffe dans les systèmes HPC.

L'étude présentée par Nie et al. [71] s'intéressent aux différents facteurs tels que la température, la consommation électrique ou encore les caractéristiques de la charge de travail pouvant augmenter la probabilité d'apparition de défaillances sur les cartes graphiques. Les données utilisées proviennent du système HPC *Titan*. En utilisant ces différents facteurs, ils proposent une méthode utilisant des algorithmes d'apprentissage machine pour tenter de prédire les défaillances survenant sur les cartes graphiques. L'algorithme d'apprentissage machine présentant les meilleurs résultats dans leur cas est le *Gradient Boosting Decision Tree*. Cet algorithme parvient à prédire 87% des défaillances avec une précision de 76%. Il est à noter qu'ils n'utilisent pas directement en entrée les données représentant les différents facteurs (la température, la consommation électrique, etc.) mais ils utilisent des données agrégées à l'aide d'opérateurs mathématiques comme la moyenne ou l'écart-type. Bien que nous utilisions aussi un algorithme d'apprentissage machine, nos travaux présentent des différences. En effet, l'utilisation d'opérateurs mathématiques pour agréger les données nécessite des évaluations préalables pour savoir quels sont les opérateurs qui permettent de mieux représenter les données. Notre solution propose d'automatiser la représentation des séries temporelles en utilisant l'algorithme de la DTW. Ce choix est réalisé en tenant compte des conclusions de deux études sur l'analyse des séries temporelles [6, 26]. De plus, dans notre cas, nous avons montré que la surchauffe des processeurs n'est pas liée des caractéristiques sur la charge de ceux-ci et que la température est le seul indicateur fiable trouvé permettant de prédire les surchauffes. Enfin, leur méthode propose de classer *à posteriori* les différentes exécutions des applications. C'est-à-dire qu'en utilisant l'ensemble des données correspondant à l'exécution d'une application, déterminer si cette exécution a subi une défaillance durant son temps d'exécution. Notre méthode propose de prédire les surchauffes *avant* qu'elles ne surviennent.

Les solutions fournies par Tuncer et al.[85] et Netti et al.[69] suivent la même idée que notre solution : analyser les métriques fournies par le système et construire un modèle d'apprentissage machine pour classer les métriques en fonction d'une défaillance. Les deux études utilisent un injecteur de défaillances pour générer 8 types de défaillances touchant différents composants : baisse de fréquence du processeur, saturation de la bande passante des disques durs, etc. Les auteurs utilisent 6 applications différentes pour mesurer l'influence des différentes erreurs injectées. L'injection de ces défaillances se déroule lors de l'exécution d'une application sur un seul nœud de calcul. Notre étude présente deux différences majeures. La première est que nous étudions des erreurs réelles provenant d'un

système HPC utilisé en production. La prise en compte des erreurs et des données de supervision réelles entraîne certaines limites, notamment en ce qui concerne la précision ou la disponibilité des données. De plus, nos travaux s'intéressent à la surchauffe des processeurs et ce problème n'est pas adressé dans leurs travaux.

Les travaux présentés par Yun et al. [103] sont les plus proches de notre étude car ils s'intéressent aussi à la prédiction des surchauffes survenant sur les processeurs. Cependant, il y a des différences significatives avec nos travaux ce qui rend l'application de leur méthode difficile dans notre cas. Leur approche générale est d'utiliser les informations à propos de la charge attendue du processeur dans un futur proche pour prédire les prochains points de température. Cette prédiction permet de lever une alarme si les prochains points de température dépassent un seuil. La solution présentée dans ce document ne requiert aucune information à propos de la charge du processeur. Même si l'obtention de ces informations est possible dans un système HPC, elle nécessiterait l'utilisation de méthodes intrusives dans le code source de l'application ou l'ajout d'une nouvelle couche interceptant les instructions passées au processeur dans le but de pouvoir caractériser et connaître la charge du processeur. Par conséquent, nous avons décidé d'exclure cette solution. En effet, comme présenté dans la section 4.1, une de nos contraintes fixées est d'avoir un système facilement embarquable et déployable ne nécessitant pas de modifications de l'existant. De plus, les résultats présentés dans le paragraphe 4.3.5 démontrent que l'utilisation de méthodes se reposant sur un seuil peut ne pas fournir des résultats fiables dans le cas de la prédiction des surchauffes de processeur avec des données d'entrée provenant d'un système HPC en production et donc peu précises. Notre solution repose sur l'analyse des tendances de la température des processeurs avant un événement de surchauffe pour être générique et pour être capable de fournir des résultats fiables avec des données d'entrée peu précises au lieu de focaliser sur des informations précises comme des variations rapides contenues dans les données.

4.2 Description et fonctionnement de la solution

Dans cette section, nous décrivons la solution proposée en détaillant chaque étape et en expliquant les deux phases différentes nécessaires à son fonctionnement : la phase d'apprentissage et la phase de test. Nous expliquons ensuite la construction du jeu de données utilisé pour entraîner et évaluer la solution.

4.2.1 Description de notre solution

La classification des séries temporelles repose sur 3 grandes étapes : la réduction de dimension, la mesure de similarité et la classification en elle-même. La figure 4.1 présente les différentes étapes de notre solution : la réduction de dimension est réalisée à l'aide de fenêtres temporelles ; la mesure de similarité est réalisée à l'aide de la DTW et la classification utilise un algorithme d'apprentissage machine. L'étape de réduction de dimension est réalisée pour réduire la taille des données en entrée. L'étape de mesure de similarité est réalisée pour utiliser des données avec une faible précision et un faible taux d'échantillonnage et pour permettre de mieux capturer les variations de température caractéristiques d'un événement de surchauffe. L'algorithme d'apprentissage machine est utilisé afin d'identifier les variations qui mènent effectivement à un événement de surchauffe. En complément de ces trois étapes, nous utilisons un module d'agrégation. Son but est

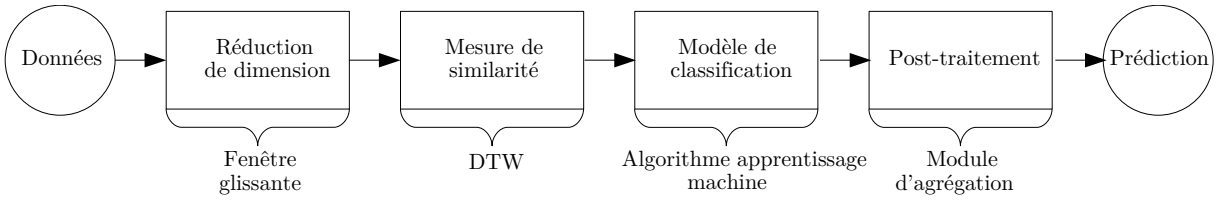


FIGURE 4.1 – Flux de traitement de notre solution.

d'analyser la sortie fournie par l'algorithme de classification en utilisant des corrélations temporelles pour améliorer la précision de la solution.

Notre système prend donc en entrée une série temporelle correspondante à la température du processeur. Cette série temporelle peut être de n'importe quelle durée. Dans une étape de prétraitement, nous générons un nouvel ensemble de petites séries temporelles de taille `L` créées en utilisant la série temporelle initiale et en faisant glisser une fenêtre sur celle-ci. Cela signifie que pour chaque processeur, à chaque nouveau pas de temps, une nouvelle série temporelle est créée incluant les valeurs de la température du processeur durant les `L` derniers pas de temps.

Nous utilisons la DTW pour mesurer la similarité entre les séries temporelles provenant de ce nouvel ensemble. Elle nous permet de calculer la distance *minimale* entre deux séries temporelles. Calculer la DTW entre séries temporelles nous permet donc de caractériser une série temporelle en la comparant à une autre. Après cette étape, les corrélations entre les distances calculées et les classes correspondantes (`surchauffe` ou `normal`) doivent être apprises. Le module de classification est responsable de cette tâche. Plus précisément, il est en charge de classer chaque petite série temporelle en ne tenant compte que des distances calculées par la DTW. Pour ce module, nous nous reposons sur l'utilisation d'un algorithme d'apprentissage machine.

Enfin, pour augmenter la précision de nos résultats, nous introduisons un module supplémentaire qui agrège les prédictions faites sur plusieurs séries temporelles consécutives. L'idée est que si les N dernières séries temporelles d'un processeur ont été classées en tant que séries temporelles menant à un événement de surchauffe par l'algorithme d'apprentissage machine, nous pouvons être plus confiant que la prédiction actuelle de surchauffe est correcte et qu'un événement de surchauffe va réellement survenir comparativement au cas où une seule série temporelle serait classée en tant que série temporelle menant à un événement de surchauffe par l'algorithme d'apprentissage machine. Le module d'agrégation considère les N dernières séries temporelles générées par un processeur et ne rapporte un événement de surchauffe que si au moins M des N dernières séries temporelles ont été classées en tant que surchauffe et que la série temporelle actuelle est classée en tant que surchauffe.

L'utilisation d'un algorithme d'apprentissage machine requiert une phase d'apprentissage pour qu'il puisse *apprendre* les corrélations entre son entrée et sa sortie. Ici, il doit apprendre les corrélations entre les mesures de similarité fournies par la DTW et les événements de surchauffe détectés dans notre système. Une fois l'apprentissage terminé, l'algorithme peut être utilisé pour classer des nouvelles séries temporelles en réalisant les mêmes mesures de similarité. Nous décrivons maintenant ces deux phases.

Phase d'apprentissage

Comme précédemment expliqué, notre solution utilise un jeu de données d'un ensemble de séries temporelles se recouvrant. Chaque série temporelle est classée en tant que

`surchauffe` ou `normale` selon qu'elle mène ou non à une surchauffe. Le classement des séries temporelles utilise la localisation temporelle des événements de surchauffe pour déterminer si une série temporelle doit être associée à la classe `surchauffe` ou `normale`. Cette localisation temporelle est fournie par les entrées associées à une surchauffe présentes dans les journaux systèmes. Le classement est détaillé dans le paragraphe 3.3.2. Un sous-ensemble de ces séries temporelles est sélectionné de manière aléatoire pour la phase d'apprentissage.

La DTW est ensuite calculée pour construire les vecteurs utilisés en entrée de notre algorithme d'apprentissage machine. Pour chaque série temporelle dans le jeu de données (classée en tant que surchauffe ou normale), nous construisons un vecteur de DTW. Pour une série temporelle donnée, son vecteur de DTW est donc composé de la valeur de la DTW entre cette série temporelle et les autres séries temporelles incluses dans le jeu de données d'apprentissage.

Ce jeu de données est fourni en entrée de notre algorithme d'apprentissage machine. Grâce à la connaissance de chaque classe des séries temporelles, nous sommes capables de fournir en entrée le vecteur de DTW et en sortie la classe correspondante. Ainsi, l'algorithme apprend les relations entre les DTW calculées et la classe de la série temporelle. Un seul modèle d'apprentissage machine est en charge de l'apprentissage des relations pour l'ensemble des séries temporelles provenant de différents processeurs.

Il est à noter que cette approche diffère des techniques de l'état de l'art décrites dans le papier proposé par Xing et Al. [99]. Dans une solution classique où un algorithme comme le kNN est utilisé, l'algorithme prend directement en entrée la série temporelle et utilise la DTW pour calculer la distance entre les séries temporelles. Dans notre solution, nous utilisons un vecteur de DTW comme entrée de l'algorithme de classification. Nous avons comparé les deux techniques en utilisant l'algorithme du kNN et nous avons obtenu des résultats similaires. Cependant, notre technique nous permet d'utiliser facilement d'autres algorithmes de classification à la place du kNN car elle ne nécessite pas de modifier le calcul de distance *interne* à l'algorithme. C'est pour cette raison que nous avons décidé de conserver notre méthode en fournissant en entrée un vecteur de DTW.

Phase de test

Lorsque le modèle est entraîné, il peut être utilisé pour classer de nouvelles séries temporelles et prédire de nouveaux événements de surchauffe survenant dans notre système. Quand une nouvelle série temporelle doit être classée, la première étape est de construire son vecteur de DTW en calculant les distances avec toutes les séries temporelles du jeu de données d'apprentissage. Ce vecteur est ensuite fourni en entrée de notre modèle d'apprentissage.

4.2.2 Création du jeu de données

Notre solution utilise un jeu de données contenant deux types de séries temporelles : des séries temporelles classées en tant que série temporelle menant à une surchauffe et des séries temporelles correspondant au comportement normal du processeur. Dans ce paragraphe, nous détaillons la création de ce jeu de données et les choix réalisés pour classer les séries temporelles en utilisant les événements de surchauffe précédemment détectés sur notre système. Nous détaillons aussi les étapes de nettoyage et de filtrage du jeu de données nous permettant d'obtenir des résultats fiables.

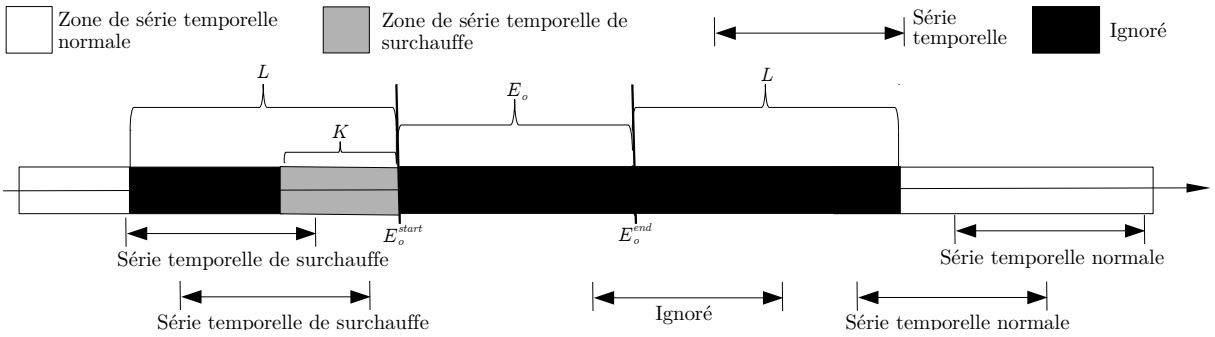


FIGURE 4.2 – Définition des classes des séries temporelles.

Classement des séries temporelles

Une fois les événements correspondants au début d'un événement de surchauffe correctement identifiés (section 3.3), nous devons classer les séries temporelles représentant la température des processeurs en fonction de ces événements. La figure 4.2 résume les différents cas pour classer une série temporelle. Nous distinguons 3 cas : les séries temporelles de surchauffe, les séries temporelles normales et les séries temporelles que nous ignorons. La classe d'une série temporelle dépend de la location temporelle de son dernier point. Si le dernier point de la série temporelle est dans une zone grise alors cette série temporelle est considérée comme une série temporelle de surchauffe, si le dernier point de la série temporelle est dans une zone blanche alors cette série temporelle est considérée comme une série temporelle normale et si le dernier point de la série temporelle est dans une zone noire alors la série temporelle est ignorée.

Ces zones sont déterminées en utilisant plus d'informations que simplement l'horodatage des événements E_o^{start}. En effet, nous utilisons les K séries temporelles précédant l'événement E_o^{start} en incluant celle contenant E_o^{start} pour fixer la largeur de la zone grise. Notre modèle est un modèle descriptif, c'est-à-dire qu'il est uniquement capable de classer la série temporelle courante en tant que série temporelle de surchauffe ou de série temporelle normale. En classant les séries temporelles précédant l'événement E_o^{start} en tant que série temporelle de surchauffe, nous nous attendons à ce que notre modèle apprenne aussi à classer les séries temporelles *avant* l'événement E_o^{start} en tant que série temporelle de surchauffe. Ce classement en *avance* permet d'obtenir un modèle prédictif.

Nous choisissons d'ignorer toutes les séries temporelles contenant une entrée L_o autre que celle qui correspond à l'événement E_o^{start}. Pour rappel, une entrée L_o est une entrée présente dans les journaux systèmes associée à un événement de surchauffe. Autrement dit, nous ignorons toutes les séries temporelles contenant tout ou une partie de l'événement de surchauffe E_o autre que son début. Ce choix est fait pour éviter que notre modèle apprenne à prédire des événements autres que ceux du début de l'événement de surchauffe E_o^{start}. Nous ignorons aussi $L - K$ séries temporelles avant E_o^{start} dans le jeu d'apprentissage. La raison est que ces séries temporelles ont des points communs avec les séries temporelles classées en tant que surchauffe et par conséquent nous voulons éviter de classer les séries temporelles proches en tant que séries temporelles normales pour ne pas classer des possibles augmentations précoces de la température en tant qu'événement normal.

Découpe du jeu de test et du jeu d'apprentissage

Une fois que notre jeu de données est construit, nous devons le découper en deux parties distinctes : le jeu de données destiné à l'apprentissage de notre modèle et le jeu de données

destiné à tester les performances de notre modèle. Le ratio entre le nombre de séries temporelles de surchauffe utilisées pour l'apprentissage et pour le test est appelé R. Notre jeu de données initial contient beaucoup plus de séries temporelles normales que de séries temporelles de surchauffe. Nous devons donc équilibrer celui-ci sous peine de voir notre modèle se focaliser uniquement sur la prédiction des séries temporelles normales. Pour effectuer cet équilibrage, nous définissons un ratio entre le nombre de séries temporelles de surchauffe et le nombre de séries temporelles normales. Ce ratio est appelé B. Ce ratio est uniquement utilisé pour construire le jeu de données d'apprentissage. Le jeu de données de test contient l'ensemble des séries temporelles non sélectionnées pour faire partie du jeu d'apprentissage.

Nettoyage et sélection des données

Les données de température utilisées sont pour la validation de notre solution fournies par le système HPC présenté dans la section 3.1. Nous utilisons ici les données de température de mars 2017 à octobre 2017 et les journaux systèmes correspondants pour la détection des événements de surchauffe.

Notre jeu de données initial contient des points de données manquants. Nous considérons donc comme valide, une série temporelle contenant au maximum 5 points de données manquants. Par conséquent, si la taille de la série temporelle est de 40 points, elle sera considérée comme valide si elle contient au minimum 35 points de données. Les séries temporelles invalides sont ignorées. Il est à noter que l'algorithme de DTW peut être utilisé pour comparer des séries temporelles avec des tailles différentes. Nous n'utilisons donc pas d'algorithme de régression pour compléter les points manquants.

Dans la pratique, il n'est pas utile de classer les séries temporelles toutes les minutes (*i.e.* à chaque nouveau point de température). En effet, il est inutile de classer les séries temporelles correspondantes à un processeur *froid*. En ajoutant une étape de filtrage avant notre solution, nous pouvons réduire grandement son coût en temps de calcul. Le critère pour définir qu'un processeur est *froid* est le suivant : c'est le minimum des températures maximales incluses dans une fenêtre temporelle lors d'une surchauffe. Le coût en temps de calcul de notre solution est discuté dans le paragraphe 4.3.6.

Au départ, notre jeu de données contient 14 641 séries temporelles différentes pour 6 mois de données. Ces séries temporelles correspondent à des processeurs considérés comme *chauds*, c'est-à-dire dont la température est supérieure au seuil déterminé précédemment pour considérer qu'un processeur est *froid*. Dans notre cas, un processeur est considéré comme *chaud* si la valeur de sa DTS est supérieure à -6. Précisons que ce seuil ne supprime aucun événement de surchauffe dans nos données. Après avoir supprimé les séries temporelles contenant plus de 5 points de données invalides, nous obtenons 14 543 séries temporelles et 85 événements de surchauffe. Par conséquent, nous avons $85 \times K$ séries temporelles qui peuvent être considérées comme des séries temporelles de surchauffe. Le reste des séries temporelles, à l'exception des séries temporelles ignorées, sont considérées en tant que séries temporelles normales. Finalement notre jeu de données d'apprentissage contient $85 \times R$ événements de surchauffe, $85 \times R \times K$ séries temporelles de surchauffe et $85 \times R \times K \times B$ séries temporelles normales. Le jeu de données de test est composé du jeu de données initial duquel l'ensemble des séries temporelles utilisées pour le jeu de données d'apprentissage est retiré. Par conséquent nos résultats de prédiction sont calculés sur les $85 \times (1 - R)$ événements de surchauffe restant.

La valeur de R est fixée à 40% pour toutes nos expériences. La valeur de B dépend des propriétés attendues de la solution : une grande valeur de B implique un plus grand nombre

de séries temporelles normales dans les données d'apprentissage et donc une focalisation sur celles-ci, une petite valeur de B implique un nombre plus faible de séries temporelles normales et donc une focalisation sur les séries temporelles de surchauffe.

4.3 Évaluation de la capacité prédictive solution

La solution étant décrite, nous pouvons maintenant procéder à l'évaluation de celle-ci. L'évaluation présentée dans cette section est une évaluation de la solution qui ne prend pas en compte d'éventuelles actions utilisant sur les prédictions. Nous présentons donc les résultats qui concernent uniquement les capacités prédictives de la solution. Nous commençons par présenter et illustrer les métriques qui nous servirons pour noter notre modèle. Ensuite, nous présentons les différents algorithmes évalués pour le classement des séries temporelles. Enfin, nous présentons la méthode pour choisir les paramètres menant aux résultats que nous présenterons dans la dernière partie.

4.3.1 Présentation et illustration des métriques

Évaluer la qualité des prédictions pour un modèle prédictif comme celui présenté dans ce chapitre est une tâche complexe. En effet, le problème peut se résumer à la question suivante : Combien de temps en avance la prédiction d'une surchauffe peut être faite sans considérer que cette prédiction est fausse ? Nous commençons par présenter les métriques classiques utilisées pour noter des modèles d'apprentissage machine, puis nous présentons nos extensions à ces métriques pour prendre en compte l'aspect temporel de notre prédiction.

Pour noter un modèle générique d'apprentissage machine, 4 classes de prédiction sont utilisées : Vrai Positif (VP, *i.e.* une prédiction d'une série temporelle de surchauffe correcte), Vrai Négatif (VN, *i.e.* une prédiction d'une série temporelle normale correcte), Faux Positif (FP, *i.e.* une prédiction d'une série temporelle de surchauffe incorrecte) et Faux Négatif (FN, *i.e.* une prédiction d'une série temporelle normale incorrecte). En utilisant ces 4 classes, nous pouvons définir les métriques de *rappel* et de *précision*. Elles sont définies comme suit :

$$Rappel = \frac{VP}{VP + FN} \quad Précision = \frac{VP}{VP + FP}$$

Le rappel note la capacité à retrouver tous les événements de surchauffe alors que la précision note la capacité à prédire correctement l'état de sortie et à ne pas se tromper lors d'une prédiction. La mesure F_1 est une combinaison des métriques précédentes pour noter de manière globale les performances de l'algorithme. Elle est définie comme suit :

$$mesure\ F_1 = \frac{2 \times Rappel \times Précision}{Rappel + Précision}$$

Ces métriques ne prennent pas en compte le fait que nous voulons noter notre modèle sur sa capacité à faire des prédictions *avant* les événements de surchauffe. Par exemple, si une prédiction est réalisée 2 minutes en avance, elle devrait être considérée comme une prédiction correcte malgré le fait que son horodatage ne soit pas le même que l'horodatage de l'événement de surchauffe. Nous introduisons donc de nouvelles métriques qui étendent les métriques précédemment présentées avec des notions temporelles. En considérant E_o^{start}

comme le début de l'événement de surchauffe et $T(E_o^{start})$ l'horodatage associé à cet événement, les métriques sont définies par :

$Ri[x_r, x_p]$ est le pourcentage d'événements de surchauffe avec au moins une série temporelle prédite en tant que surchauffe dans l'intervalle [$T(E_o^{start})$ - x_p, $T(E_o^{start})$ - x_r].

$P[x_p]$ est le pourcentage de séries temporelles de surchauffe correctement prédites. Une prédiction est considérée comme correcte si elle est dans l'intervalle [$T(E_o^{start})$ - x_p, $T(E_o^{start})$].

$F_1[x_r, x_p]$ est la combinaison de $Ri[x_r, x_p]$ et $P[x_p]$ pour permettre de noter globalement le modèle. $F_1[x_r, x_p] = \frac{2 \times Ri[x_r, x_p] \times P[x_p]}{Ri[x_r, x_p] + P[x_p]}$

La métrique $Ri[x_r, x_p]$ permet de noter la capacité de notre modèle à prédire les événements de surchauffe et la métrique $P[x_p]$ permet de noter la précision de notre modèle, c'est à dire sa capacité à éviter les mauvaises classifications.

La valeur x_r correspond au délai minimal avant un événement de surchauffe pour considérer une prédiction comme utile et la valeur x_p correspond au délai maximal avant un événement de surchauffe pour considérer que la prédiction est correcte. Les valeurs de x_p et x_r sont choisies en prenant en compte les caractéristiques d'une action préventive qui serait exécutée à partir des prédictions réalisées. Avec ces paramètres, nous supposons que cette action générique nécessitera au minimum x_r minutes pour être efficace et que l'efficacité durera au maximum x_p minutes.

Dans cette première partie de l'évaluation où nous cherchons à évaluer les capacités prédictives de notre solution, la valeur de x_p est sélectionnée en fonction des variations de la température avant un événement de surchauffe étudiées dans le paragraphe 3.4.3. Nous pouvons voir que l'augmentation concernant le percentile correspondant au 20% des températures commence à partir de 40 minutes avant un événement de surchauffe. Nous pensons qu'il est acceptable de considérer qu'une prédiction réalisée plus de 40 minutes en avance soit considérée comme un faux positif. Nous fixons donc $x_p = 40\ minutes$.

Illustration des métriques

Nous utilisons la figure 4.3 pour montrer un exemple de calcul des métriques précédemment introduites. Sur cette figure, les carrés situés au-dessus de la ligne indiquent la classe réelle de chaque pas temporel et les carrés situés en dessous de la ligne indiquent la classe prédite de chaque pas de temps. Chaque pas de temps correspond au dernier point d'une série temporelle. Ainsi, les vrais positifs et vrais négatifs correspondent au cas où la couleur d'en dessous de la ligne est la même que la couleur d'au-dessus de la ligne. Les faux positifs et les faux négatifs correspondent au cas où les deux carrés sont de couleurs différentes. Par exemple, un pas de temps avec une couleur blanche en haut et une couleur noire en haut indique que notre solution prédit un événement de surchauffe mais que l'état réel du processeur est normal. C'est donc un faux positif.

Dans cette figure, nous décrivons 3 événements de surchauffe E_{o1}^{start}, E_{o2}^{start} et E_{o3}^{start}. Pour cet exemple, nous supposons qu'une prédiction est correcte si elle faite au plus 6 pas de tempss avant un événement de surchauffe, ce qui revient à fixer $x_p = 6$. De la même manière, nous supposons qu'une prédiction doit être réalisée au minimum 2 pas de temps avant un événement de surchauffe pour être utile. Nous fixons donc $x_r = 2$. Les prédictions réalisées dans les zones noires sont ignorées car elles correspondent aux zones ignorées lors de l'étape d'apprentissage (le paragraphe 4.2.2 détaille la construction de ces zones). Seuls les événements E_{o1}^{start} et E_{o2}^{start} ont une prédiction correcte dans l'intervalle [$T(E_o^{start})$ - 6,

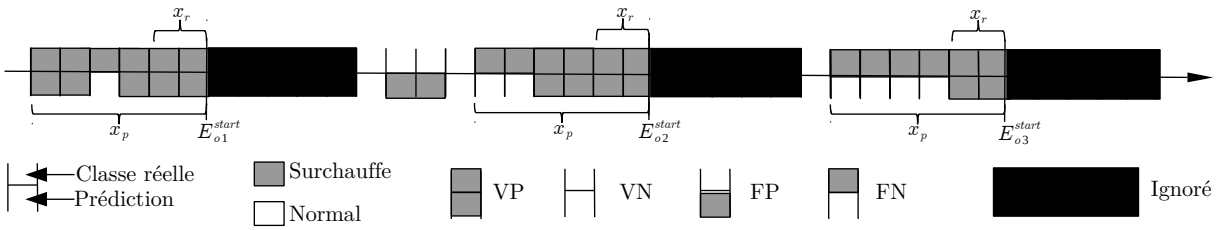


FIGURE 4.3 – Exemple de prédictions.

$T(E_o^{start})$ - 2]. Par conséquent, seulement 2 événements sur 3 sont correctement prédits par notre solution dans cet exemple. Nous pouvons donc calculer Ri qui correspond au rappel de l'intervalle $[-6, -2]$:

$$Ri[x_r = 2, x_p = 6] = \frac{2}{3} = 66\%$$

En utilisant le nombre de vrais positifs et de faux positifs, nous pouvons calculer P :

$$P[x_p = 6] = \frac{VP}{VP + FP} = \frac{11}{11 + 2} = 85\%$$

En d'autres termes, dans cet exemple, nous sommes capables de prédire 66% des événements de surchauffe au moins 2 minutes en avance et 85% des prédictions de surchauffe sont situées dans les 6 minutes qui précèdent un événement de surchauffe.

4.3.2 Algorithmes évalués pour le classement des séries temporelles

Notre solution utilise un algorithme de classification pour le classement des séries temporelles en deux classes : celles menant à une surchauffe et celles menant à un état normal. Nous évaluons ici 4 algorithmes d'apprentissage machine.

En complément de ces algorithmes, nous évaluons aussi une approche alternative. Cette approche se focalise sur la prédiction des prochaines valeurs de la température des processeurs. Si les prédictions des prochaines valeurs de la température du processeur dépassent un seuil, une prédiction d'un événement de surchauffe est réalisée. Nous évaluons aussi la pertinence d'un seuil seul sans prédiction des prochaines valeurs de la température des processeurs.

Dans un premier temps, nous allons décrire les algorithmes d'apprentissage machine utilisés pour nous décrirons les algorithmes utilisant des seuils.

Algorithmes d'apprentissage machine

Le choix des algorithmes d'apprentissage machine est dicté par les caractéristiques de nos données et par les objectifs que nous nous sommes fixés. Ainsi, nous voulons un algorithme capable d'apprendre en utilisant un faible nombre d'exemples (moins de 100 dans notre cas) sans *sur-apprendre*, c'est-à-dire en gardant une capacité de généralisation à des exemples non vus. En tenant compte de ces contraintes, nous avons identifié 4 algorithmes d'apprentissage machine : l'algorithme du *Gradient Boosting Tree* (GBT), l'algorithme des k plus proches voisins (kNN), l'algorithme de l'arbre de décision et l'algorithme de la forêt aléatoire. Les réseaux de neurones et les techniques d'apprentissage profond sont exclus à cause du faible nombre d'exemples. Nous avons aussi testé l'algorithme du séparateur à

vaste marge (SVM). Cependant, il n'apporte pas d'amélioration par rapport aux autres solutions, nous avons donc décidé de ne pas l'inclure dans la suite pour simplifier la lecture.

Les algorithmes d'apprentissage machine sont détaillés dans la section 2.3. Nous en faisons ici un bref résumé :

— L'algorithme du kNN effectue un vote parmi les k plus proches voisins. Si la majorité des plus proches voisins appartient à la classe `A`, alors la classe du point actuel sera `A`. Cet algorithme dépend fortement de la distribution des données.

— L'algorithme de l'arbre de décision détermine des conditions de décision sur les données d'entrée.

— L'algorithme de la forêt aléatoire construit plusieurs arbres de décision au lieu d'un seul. Cela permet d'éviter les problèmes de sur-apprentissage.

— L'algorithme du GBT construit plusieurs petits arbres. Chaque nouvel arbre utilise les résultats de l'arbre précédent. Cela permet d'augmenter la capacité de généralisation du modèle.

Algorithmes utilisant un seuil

Pour évaluer la performance de notre solution utilisant des algorithmes d'apprentissage machine, nous évaluons aussi une autre approche qui utilise des méthodes de prédiction faisant appel à la définition un seuil. En effet, nous voulons évaluer si un simple seuil sur la température du processeur est capable de prédire efficacement le prochain événement de surchauffe. La prédiction de surchauffe sera réalisée en utilisant directement la température du processeur. Si la température du processeur est supérieure à un seuil alors la solution prédira une surchauffe sur celui-ci.

En complément de cette première méthode, nous évaluons une deuxième méthode incluant un oracle précédant le seuil. L'oracle a la capacité de prédire *parfaitement* (*i.e.* sans erreur) les prochains points de température. Dans la pratique, nous utilisons les points de température disponibles dans le jeu de données pour réaliser la prédiction. Cet oracle nous permet de borner la performance maximale théorique d'une solution utilisant un modèle pour prédire les prochaines valeurs de la température (par exemple ARIMA [97], les réseaux de neurones récurrents [48], les modèles de régression linéaire [102], etc.) sans l'évaluer directement. En effet, si ces modèles de prédiction de températures sont parfaits, ils réaliseront des prédictions parfaites ce qui correspondra à la performance de notre oracle. Dans le cas contraire, les erreurs faites par les solutions de prédiction de température auront une influence, en générale négative, sur les résultats globaux de la solution de prédiction des surchauffes.

Comme expliqué dans le paragraphe 4.1.2, l'utilisation de cet oracle nous permet de comparer deux méthodes pour la prédiction de la surchauffe : une première méthode utilisant l'analyse de la température actuelle pour la prédiction des surchauffes et une deuxième utilisant la prédiction de la température dans le futur pour être capable de détecter les surchauffes. Il est à noter que ces deux solutions ne sont pas antagonistes, il est envisageable de réaliser d'abord une prédiction de la température, puis une analyse de cette prédiction pour détecter les signes précurseurs de la surchauffe. Cette solution est discutée dans la section 4.5.

4.3.3 Sélection des paramètres

Notre solution utilisant des algorithmes d'apprentissage machine ainsi que les solutions utilisant des seuils requièrent de fixer la valeur de plusieurs paramètres. Les tableaux 4.2 et 4.3 présentent les paramètres utilisés lors de nos expériences pour optimiser la mesure F_1 (section 4.3) et pour optimiser la réduction du coût des surchauffes (section 4.4). Le tableau 4.2 présente également les intervalles de variation de ces paramètres.

La sélection des paramètres est réalisée en testant la combinaison de l'ensemble des paramètres 5 fois. Nous sélectionnons ensuite le meilleur ensemble de paramètres en fonction de la métrique que nous souhaitons optimiser.

Pour le seuil, les valeurs dans l'intervalle [-10, 0] sont testées avec un pas de 1 degré.

En plus de ces paramètres, chaque algorithme d'apprentissage machine dispose de paramètres internes qui peuvent être modifiés. Le but de cette étude n'est pas d'optimiser chaque algorithme d'apprentissage machine pour une tâche précise. Nous voulons évaluer quel est le type d'algorithme permettant d'obtenir les meilleurs résultats dans un maximum de cas sans chercher à évaluer sa performance maximale dans chacun des cas. Cela nous permet d'avoir une solution générique capable de fournir des résultats fiables dans une grande variété de cas. Les paramètres internes sont donc laissés sur les paramètres par défaut. Ces paramètres sont présents dans la documentation du cadriciel utilisé [1].

4.3.4 Choix techniques et cadriciels

Nous avons utilisé le langage Python dans sa version 3.6 pour le développement de la solution. Nous avons utilisé deux cadriciels : scikit-learn [2] pour les algorithmes d'apprentissage machine et l'implémentation de la DTW fournie par DTAI Research Group [3].

4.3.5 Résultats

Nous allons maintenant présenter les résultats obtenus par les deux approches : celle de caractériser les signes précurseurs pour prédire les surchauffes à l'aide d'algorithme d'apprentissage machine et celle de prédire les prochaines valeurs de la température et de prédire les surchauffe à l'aide de seuil. Nous évaluons aussi la performance de différents algorithmes d'apprentissage machine pour la caractérisation des signes précurseurs.

Comme expliqué précédemment, pour la suite des résultats présentés dans cette section nous fixons la valeur de $x_p = 40min$ en utilisant les caractéristiques de l'augmentation des températures avant la surchauffe. Nous devons maintenant décider quelle sera la valeur de x_r pour calculer les métriques Ri et F_1. Nous rappelons que dans cette partie, nous voulons évaluer notre solution sans tenir compte d'une action préventive spécifique. Pour être capable d'analyser un large ensemble d'actions préventives possibles, nous décidons d'analyser nos résultats en prenant en compte trois valeurs de x_r : 1 minute, 5 minutes et 10 minutes. En d'autres termes, nous voulons évaluer notre solution sur sa capacité à prédire les événements de surchauffe avec 1 minute, 5 minutes et 10 minutes d'avance. Par conséquent, pour chaque valeur de x_r, nous voulons optimiser respectivement $F_1[1, 40]$, $F_1[5, 40]$ et $[10, 40]$. Six solutions sont évaluées dans nos tests : la méthode utilisant un seuil, la méthode utilisant un seuil avec un oracle, notre solution utilisant le GBT, notre

1. https ://scikit-learn.org/stable/index.html
2. https ://scikit-learn.org/stable/index.html version 0.20.2
3. https ://pypi.org/project/dtaidistance/ version 0.1.5

Description des paramètres	Intervalle des valeurs testées	Notre solution utilisant le GBT (maximisant la mesure F_1[1,40])	Notre solution utilisant le GBT (maximisant la mesure F_1[5,40])	Notre solution utilisant le GBT (maximisant la mesure F_1[10,40])	Méthodes utilisant un seuil
Taille de la série temporelle (noté L)	[20min, 120min] pas de 10min	40min	40min	60min	-
Probabilité minimum d'être classé en tant que série temporelle de surchauffe	[0.1, 1] pas de 0.1	0.9	0.8	0.6	-
Nombre de séries temporelles à prendre en compte pour le module d'agrégation	[0,120] pas de 1	6	9	7	0
Nombre minimal requis de séries temporelles précédemment prédites en tant que surchauffe pour prédire une surchauffe	[0,120] pas de 1	3	8	2	-
Température maximale pour prédire un état normal automatiquement	-	-6	-6	-6	-
Ratio entre le jeu de données d'apprentissage et de test (noté R)	-	0.4	0.4	0.4	-
Ratio entre les séries temporelles de surchauffe et normales pour le jeu de données d'apprentissage (noté B)	[2,16] pas de 2	4	2	8	-
Nombre de séries temporelles classées en tant que surchauffe avant un événement E_o^{start} dans le jeu de données d'apprentissage (noté K)	[2,16] pas de 2	1	4	8	-
Température minimale pour être classé en tant que série temporelle de surchauffe	[0,-10] pas de 1	-	-	-	-5 (seul), -4 (oracle)

Tableau 4.2 – Paramètres et leurs intervalles de test utilisés pour nos expérimentations concernant la mesure F_1. Le symbole "-" signifie que la valeur n'est pas définie pour la solution.

Description des paramètres	Notre solution utilisant le GBT (maximisant le coût de réduction T_{max} et la réduction de fréquence)	Notre solution utilisant le GBT (maximisant le coût de réduction T_{min} et la réduction de fréquence)	Notre solution utilisant le GBT (maximisant le coût de réduction T_{max} et la migration de tâches)	Notre solution utilisant le GBT (maximisant le coût de réduction T_{min} et la migration de tâches)	Méthodes utilisant un seuil
Taille de la série temporelle glissante (noté L)	120min	120min	60min	40min	-
Probabilité minimum d'être classé en tant que série temporelle de surchauffe	0.5	0.9	0.9	0.5	-
Nombre de séries temporelles à prendre en compte pour le module d'agrégation	6	2	33	53	0
Nombre minimal requis de séries temporelles précédemment prédites en tant que surchauffe pour prédire une surchauffe	1	1	1	51	-
Température maximale pour prédire un état normal automatiquement	-6	-6	-6	-6	-
Ratio entre le jeu de données d'apprentissage et de test (noté R)	0.4	0.4	0.4	0.4	-
Ratio entre les séries temporelles de surchauffe et normales pour le jeu de données d'apprentissage (noté B)	4	16	4	8	-
Nombre de séries temporelles classées en tant que surchauffe avant un événement E_o^{start} dans le jeu de données d'apprentissage (noté K)	4	4	2	2	-
Température minimale pour être classé en tant que série temporelle de surchauffe	-	-	-	-	-5 (seul), -4 (oracle)

Tableau 4.3 – Paramètres utilisés pour nos expérimentations concernant la réduction du coût des surchauffes. Le symbole "-" signifie que la valeur n'est pas définie pour la solution.

solution utilisant un arbre de décision, notre solution utilisant une forêt aléatoire et notre solution utilisant un kNN. Les résultats présentés sont obtenus en maximisant la mesure F_1 avec différentes valeurs de x_r.

Résultats des algorithmes d'apprentissage machine

Le tableau 4.4 résume les résultats obtenus en considérant les métriques $P[40]$, $Ri[x_r = \{1, 5, 10\}, 40]$ et $F_1[x_r = \{1, 5, 10\}, 40]$ pour les solutions utilisant un modèle d'apprentissage machine en optimisant les solutions en considérant x_r=1 (tableau 4.4a), x_r=5 (tableau 4.4b) et x_r=10 (tableau 4.4c). Les valeurs en gras sont les meilleurs résultats en considérant toutes les solutions pour une optimisation donnée (*i.e.* optimiser $F_1[x_r, 40]$ pour une valeur de x_r donnée). Les valeurs soulignées sont les meilleurs résultats pour une solution donnée et pour une optimisation donnée (*i.e.* optimiser $F_1[x_r, 40]$ pour une valeur x_r et une solution donnée). Le tableau fournit aussi le nombre de Faux Positifs (FP) et le nombre de Vrais Positifs (VP) pour chaque configuration. Le nombre de Faux Positifs et le nombre de Vrais Positifs sont utilisés pour le calcul du coût pratique de notre solution dans la section 4.4.

Nous pouvons voir le GBT est plus performant que tous les autres algorithmes d'apprentissage machine en considérant la mesure F_1.

De plus, nous voyons que l'utilisation d'algorithme d'apprentissage machine permet d'optimiser la solution pour chaque but souhaité. En effet, chaque nouvel apprentissage avec une nouvelle valeur métrique à optimiser permet d'être plus performant que l'ancienne optimisation sur la nouvelle valeur. Par exemple, l'optimisation de notre solution avec le GBT en considérant la valeur de la mesure $F_1[1, 40]$ nous permet d'avoir $F_1[5, 40] = 0.68$. Si un nouvel apprentissage est réalisé en optimisant la valeur de la mesure $F_1[5, 40]$ cette fois, cela nous permet d'augmenter la valeur de la mesure $F_1[5, 40]$ et d'avoir $F_1[5, 40] = 0.76$. Cela montre que notre solution est adaptable selon le but et les caractéristiques recherchées.

Il est à noter que nous pouvons aussi voir que les résultats obtenus avec un algorithme donné peuvent optimiser plusieurs mesures F_1 : les résultats obtenus par l'algorithme du kNN sont les mêmes pour $F_1[10, 40]$ et $F_1[5, 40]$ ainsi que ceux obtenus par l'algorithme de la forêt aléatoire pour $F_1[10, 40]$ et $F_1[5, 40]$.

Pour mieux comprendre les résultats de chaque algorithme, nous choisissons d'analyser plus en profondeur les résultats obtenus en optimisant $F_1[5, 40]$.

La figure 4.4 présente les résultats pour la métrique $P[x_p]$ et pour la métrique $Ri[x_r, x_p]$ obtenus à partir des modèles optimisation $F_1[5, 40]$. La figure 4.4a montre la variation de la précision $P[x_p]$ en fonction de la valeur de x_p, (*i.e.* le temps maximal entre la prédiction et l'événement de surchauffe pour que la prédiction soit considérée comme une prédiction correcte). La figure 4.4b montre la variation du rappel $Ri[x_r, x_p = 40min]$ en fonction de la valeur de x_r (*i.e.* le temps minimal entre la prédiction et l'événement de surchauffe pour que la prédiction soit utile) avec $x_p = 40min$. Pour faciliter la compréhension, les figures 4.4a et 4.4b font apparaître les valeurs de -x_p et -x_r. La valeur de $-x_p$ et de $-x_r$ correspondent à $P[x_p]$ et $Ri[x_r, x_p]$ et non $P[-x_p]$ et $Ri[-x_r, -x_p]$. Ainsi, une valeur de x_p = -20 sur la figure 4.4a indique que l'intervalle considéré est $[T(E_o^{start})$ - 20, $T(E_o^{start})]$ et que la précision correspondante est $P[x_p = 20]$

Dans la figure 4.4a, nous voyons que la valeur de la précision $P[x_p]$ décroît lorsque la valeur de x_p décroît (et donc celle de -x_p croît) car la taille de l'intervalle $[T(E_o^{start})$ - x_p, $T(E_o^{start})]$ où la prédiction est considérée comme correcte décroît lorsque x_p décroît. Dans la figure 4.4b, nous voyons que la valeur du rappel $Ri[x_r, x_p]$ augmente quand x_r décroît (et donc celle de -x_r croît) et que x_p est fixe. Ceci est dû au fait que la taille de l'intervalle

Algorithme	$P[40]$	$Ri[1,40]$	$F_1[1,40]$	$Ri[5,40]$	$F_1[5,40]$	$Ri[10,40]$	$F_1[10,40]$	VP	FP
GBT	86%	94%	**0.9**	56%	0.68	21%	0.35	256	42
kNN	62%	91%	0.74	62%	0.62	40%	0.49	70	42
DT	63%	89%	0.74	59%	0.61	35%	0.45	133	78
RND	72%	90%	0.8	56%	0.63	23%	0.35	114	44

(a) $x_r = 1$

GBT	76%	88%	0.81	76%	**0.76**	44%	0.55	92	29
kNN	63%	86%	0.72	85%	0.72	67%	0.64	318	187
DT	71%	77%	0.73	72%	0.71	51%	0.59	116	47
RND	63%	82%	0.71	79%	0.7	60%	0.61	214	125

(b) $x_r = 1$

GBT	64%	79%	0.71	70%	0.67	66%	**0.65**	77	43
kNN	63%	86%	0.72	85%	0.72	67%	0.64	318	187
DT	58%	79%	0.67	76%	0.66	73%	0.65	95	68
RND	63%	82%	0.71	79%	0.7	60%	0.61	214	125

(c) $x_r = 10$

Tableau 4.4 – Résultats obtenus en maximisant la mesure F_1 pour différentes valeurs de x_r.

$[T(E_o^{start})$ - x_p, $T(E_o^{start})$ - $x_r]$ où la prédiction est considérée comme utile augmente quand x_r décroît et que x_p est fixe.

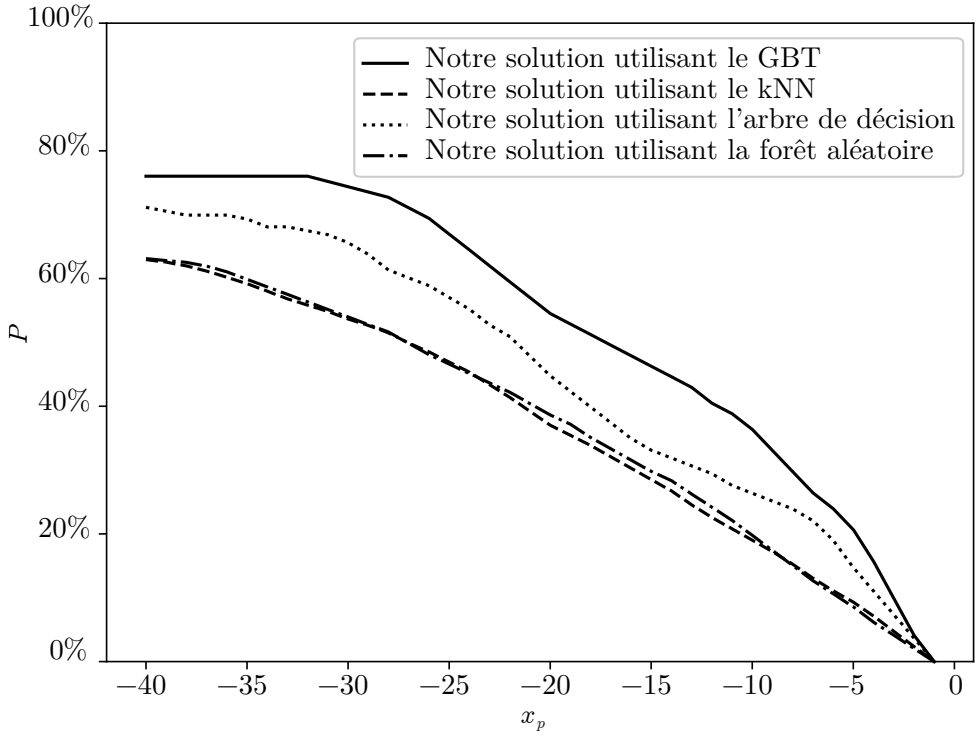
Pour analyser ces résultats, nous considérons le cas où $x_p = 40$ *minutes* et $x_r = 5$ *minutes*. En utilisant notre solution avec le GBT, nous obtenons $P[40] = 0.76$ et $Ri[5, 40] = 0.76$. En d'autres mots, 76% des prédictions de surchauffe sont dans l'intervalle $[T(E_o^{start})$ - $40min$, $T(E_o^{start})]$ et 76% des événements de surchauffe ont au moins une prédiction dans l'intervalle $[T(E_o^{start})$ - $40min$, $T(E_o^{start})$ - $5min]$.

Dans la figure 4.4a, nous voyons que notre solution utilisant le GBT a toujours une meilleure précision que les autres méthodes. Dans la figure 4.4b, nous voyons aussi que notre solution utilisant le GBT a un rappel plus faible de $x_r = 40$ à $x_r = 15$. Mais de $x_r = 15$ et $x_r = 0$, le rappel augmente rapidement et dépasse les autres méthodes à partir de $x_r = 3$. Cela s'explique car la méthode est plus sélective et à tendance à classer plus tardivement les séries temporelles (valeur de Ri augmentant tardivement) en surchauffe ce qui entraîne moins de Faux Positifs en dehors de l'intervalle considéré comme juste pour la prédiction et donc une meilleure précision.

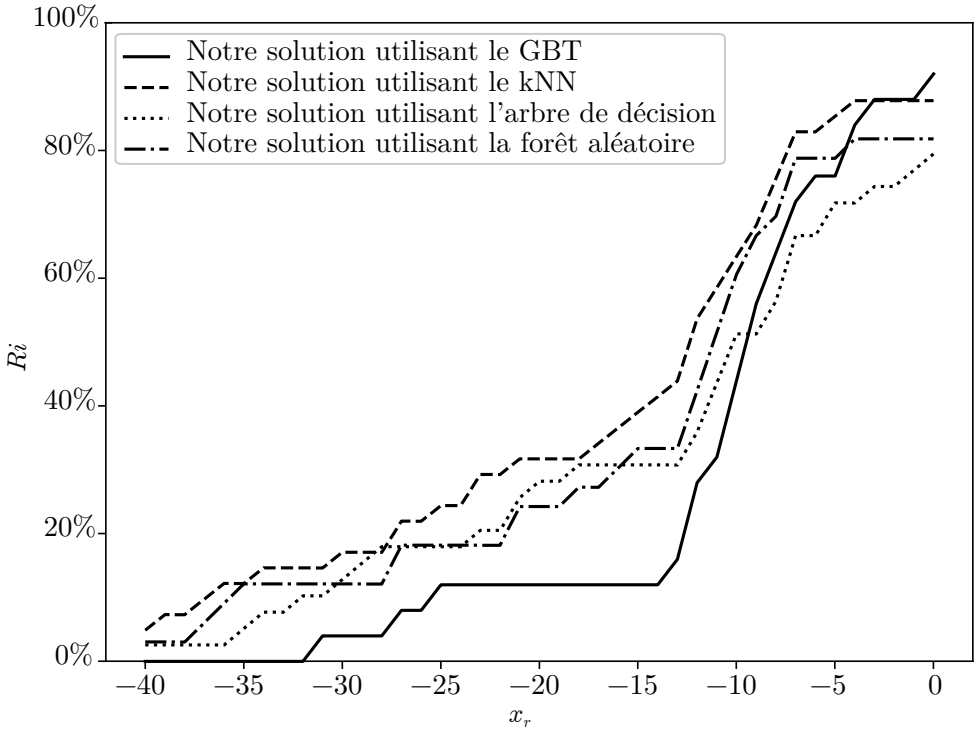
Nous pouvons donc conclure que l'algorithme du GBT nous permet d'obtenir les meilleurs résultats dans tous les cas étudiés. Dans la suite, nous choisissons donc de ne présenter que les résultats de celui-ci.

Résultats des méthodes utilisant un seuil

Nous souhaitons ici évaluer les résultats des méthodes utilisant un seuil. Ainsi, nous évaluons deux méthodes. La première méthode utilise uniquement un seuil alors que la



(a) Précision ($P[x_p]$)



(b) Rappel ($Ri[x_r, x_p = 40min]$)

FIGURE 4.4 – Évaluation du rappel et de la précision des méthodes utilisant un algorithme d'apprentissage machine.

deuxième méthode utilise un oracle en plus du seuil. De plus, nous comparons aussi les résultats obtenus avec ceux obtenus par les solutions utilisant un algorithme d'apprentissage machine.

Il est à noter que dans le tableau 4.3 les paramètres associés avec le module d'agrégation

pour les méthodes utilisant un seuil est fixé à 0. Cela signifie que pour ces méthodes, les meilleurs résultats sont obtenus quand le module d'agrégation n'est pas utilisé.

La température maximisant la mesure F_1 du seuil seul est de -5. La température maximisant la mesure F_1 du seuil avec un oracle est de -4. Il est à noter que la même valeur de seuil optimise $F_1[1, 40]$, $F_1[5, 40]$ et $F_1[10, 40]$. Par conséquent, nous présentons uniquement les résultats qui correspondent à l'optimisation de la valeur de $F_1[5, 40]$. Nous fixons la valeur de l'oracle à 5 minutes. Cela nous semble un temps raisonnable pour un algorithme de prédiction des températures.

Pour analyser les résultats des méthodes utilisant un seuil, nous procédons de la même manière que pour les solutions utilisant un algorithme d'apprentissage machine.

La figure 4.5 présente les résultats pour la métrique $P[x_p]$ et pour la métrique $Ri[x_r, x_p]$. La figure 4.5a montre la variation de précision $P[x_p]$ en fonction de la valeur de x_p, La figure 4.5b montre la variation du rappel $Ri[x_r, x_p = 40min]$ en fonction de la valeur de x_r avec $x_p = 40min$. Pour une comparaison plus facile avec les résultats précédant, nous incluons aussi les résultats de notre solution utilisant l'algorithme du GBT.

La figure 4.5a montre que la méthode utilisant un seuil seul a une précision plus faible que celle utilisant un seuil avec un oracle. Mais la figure 4.5b montre que la méthode utilisant un seuil seul a un rappel plus élevé que celle utilisant un seuil avec un oracle.

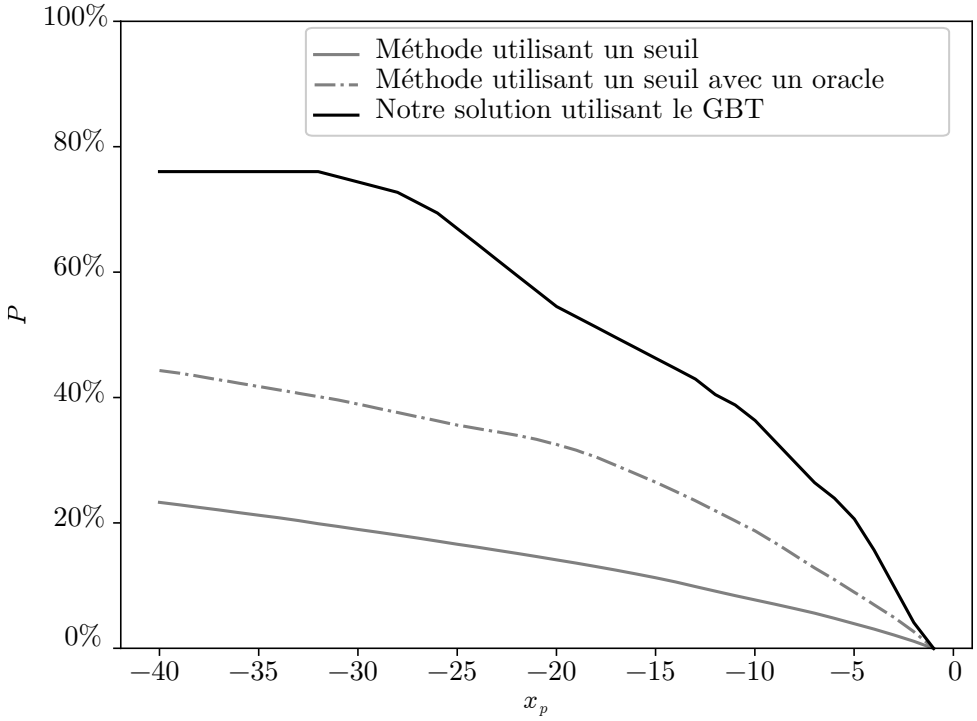
Pour mieux comprendre ce comportement, le tableau 4.5 présente le nombre de Faux Positifs et de Vrais Positifs des deux méthodes et les résultats des méthodes pour les métriques $Ri[x_r, x_p]$, $P[x_p]$ et $F_1[x_r, x_p]$ en optimisant $F_1[5, 40]$. La différence de la valeur de la température du seuil entre les méthodes (-5 pour le seuil seul et -4 pour le seuil avec un oracle) amène la méthode utilisant un oracle à être plus sélective et à réduire le nombre total de prédictions de surchauffe (1078 prédictions pour la méthode avec oracle contre 4284 prédictions pour la méthode avec un seuil seul). Cette réduction du nombre de prédictions de surchauffe induit un nombre moins élevé d'événements de surchauffe correctement prédits (valeur de Ri) mais permet d'éviter un grand nombre de Faux Positifs (influant sur la valeur de P).

Pour comprendre pourquoi la méthode utilisant un seuil seul a un rappel plus élevé mais une précision plus faible que les solutions utilisant sur des algorithmes d'apprentissage machine, nous devons analyser plus précisément les résultats. Selon la figure 4.5b 40 minutes avant un événement de surchauffe, la méthode utilisant un seuil seul classe 22% des séries temporelles en tant que surchauffe (valeur de $Ri[x_r = 40, x_p = 40]$. Cela conduit cette méthode a avoir une forte probabilité de générer un Faux Positif et donc de prédire une grande quantité d'événements de surchauffe. D'un autre côté, notre solution utilisant des algorithmes d'apprentissage machine a une très faible probabilité de générer des Faux Positifs plus de 40 minutes avant un événement de surchauffe (moins de 2% en utilisant le GBT).

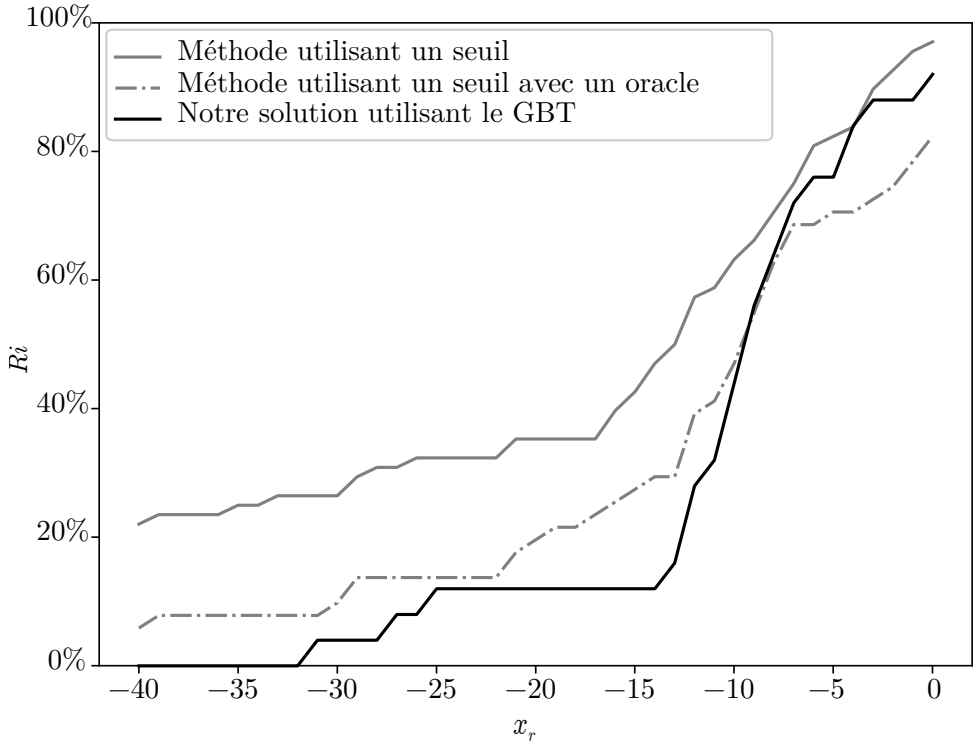
Nous pouvons donc conclure que notre solution utilisant des algorithmes d'apprentissage machine et la DTW est capable d'obtenir de meilleurs résultats que les méthodes utilisant un seuil même en considérant l'utilisation d'un oracle.

4.3.6 Coût pratique de notre système

Pour que la solution soit capable de réduire le coût des surchauffes dans le système, son surcoût pour le système doit être le plus faible possible. Nous avons évalué que le temps total mis par notre solution pour effectuer une prédiction à partir des données de température est inférieur à 3 millisecondes en utilisant un cœur de processeur et sans optimisation



(a) Précision ($P[x_p]$)



(b) Recall ($Ri[x_r, x_p = 40min]$)

FIGURE 4.5 – Évaluation du rappel et de la précision des méthodes utilisant un seuil.

particulière. De plus, nous rappelons que notre système ne réalise une prédiction que si la valeur de la DTS est supérieure à un seuil (-6 dans notre cas). Grâce à cette optimisation, la plupart du temps, aucune prédiction n'est nécessaire. Plus précisément, cela nous permet d'ignorer plus de 99% des fenêtres créées. Par conséquent, le coût de notre solution en considérant les 6 mois de données sur le système HPC de DKRZ et en considérant que

Méthode	$P[40]$	$Ri[1,40]$	$F_1[1,40]$	$Ri[5,40]$	$F_1[5,40]$	$Ri[10,40]$	$F_1[10,40]$	VP	FP
Seuil seul	25%	96%	0.40	82%	0.38	63%	0.36	1071	3213
Seuil utilisant un oracle	44%	88%	0.59	77%	0.56	43%	0.43	331	747

Tableau 4.5 – Résultats pour les méthodes utilisant un seuil en optimisant la mesure F_1 en considérant $x_p = 5$.

la valeur de la DTS est reportée toutes les minutes est inférieur à 0.16 microsecondes d'exécution par processeur et par heure.

4.3.7 Synthèse

Nous avons vu à travers cette évaluation que notre solution est capable de prédire une majorité des surchauffes tout en gardant une précision satisfaisante. L'évaluation de la qualité des prédictions de notre solution que nous sommes capables de prédire 76% des surchauffes avec 5 minutes d'avance. De plus, notre solution utilisant un algorithme d'apprentissage machine est paramétrable. Nous pouvons optimiser ses résultats selon le but envisagé. Enfin, nous avons montré que les solutions utilisant des seuils ne sont pas capables d'obtenir des résultats équivalents à solution.Dans la prochaine partie, nous allons évaluer si ces résultats sont suffisants pour espérer pouvoir réduire le coût de la surchauffe dans notre système en appliquant des actions préventives.

4.4 Analyse pratique de la solution en incluant les actions préventives

En utilisant les prédictions précédemment faites, une action préventive doit être exécutée pour permettre d'éviter les événements de surchauffe. Différentes actions préventives peuvent être envisagées. Nous en étudions deux dans cette section : une légère baisse de fréquence appliquée en avance et une migration de tâches. Nous commençons par introduire les métriques que nous utilisons pour noter notre solution en prenant en compte le coût et l'efficacité des actions préventives. Puis, nous détaillons la méthode employée pour calculer le coût de la surchauffe dans notre système. Nous présentons ensuite les actions, les coûts associés et l'évaluation de leur possible réduction du coût des surchauffes. Enfin, nous concluons sur l'utilité ou non de la solution globale.

4.4.1 Présentation et illustration des métriques

L'évaluation que nous réalisons doit nous permettre de mesurer si la capacité prédictive de notre solution est suffisante pour envisager une action par rapport à laisser les événements de surchauffe se produire dans le système. En d'autres mots, nous devons évaluer d'une part le coût total de notre solution en incluant le coût des actions prises et d'une autre part le coût total des événements de surchauffe dans le système HPC étudié.

Pour mesurer ce bénéfice, nous introduisons une métrique de réduction des coûts (notée CS par la suite). En considérant CA, le coût de notre solution incluant le coût des actions

et CO, le coût des événements de surchauffe dans notre système, CS est défini par :

$$CS = 1 - \frac{CA}{CO}$$

Si la valeur de CS est positive, alors la mise en place d'actions préventives utilisant les prédictions réalisées par notre solution peut réduire le coût des événements de surchauffe sur le système. Si la valeur de CS est négative, il est plus coûteux d'essayer de prendre des mesures préventives que de simplement laisser les événements de surchauffe se produire.

Pour calculer la valeur de CS, nous devons donc déterminer la valeur de CO et celle de CA.

La valeur de CA est définie comme la somme du coût de toutes les actions préventives réalisées et du coût des événements de surchauffe non prédits (noté CO_m). Le coût de toutes les actions préventives réalisées est le coût d'une action (noté C_a) multiplié par le nombre d'actions (noté NA). Pour calculer le coût des événements de surchauffe manqués, nous utilisons la métrique introduite dans le paragraphe 4.3.1, $Ri[x_r, x_p]$. Nous avons $CO_m = (1 - Ri[x_r, x_p]) \times CO$ où $(1 - Ri[x_r, x_p])$ correspond aux événements de surchauffe non prédits et CO au coût total des événements de surchauffe. Pour calculer la valeur de NA, nous ajoutons simplement les Faux Positifs avec les Vrais Positifs. Par conséquent, nous avons $NA = FP + VP$. Finalement, nous avons :

$$CS = 1 - \frac{CA}{CO} = 1 - \frac{(FP + VP) \times C_a + (1 - Ri[x_r, x_p]) \times CO}{CO}$$

Il est à noter que le temps nécessaire à la solution pour réaliser les prédictions sur l'ensemble de la période considérée représente moins de 1% du temps total des événements de surchauffe. De plus, sur le système HPC considéré dans ce cas d'étude, chaque nœud de calcul dispose d'un coprocesseur dédié à sa surveillance. Ce co-processeur peut être utilisé pour réaliser les prédictions de surchauffe grâce à la légèreté de calcul de la solution. Nous avons donc décidé de ne pas inclure le temps d'exécution de la solution dans nos calculs car il apparaît comme négligeable.

En utilisant l'exemple présenté dans la figure 4.3, nous montrons comment est calculé CS sur cet exemple. Pour le calculer, nous considérons arbitrairement que le coût de chaque action est de 0.01 ($C_a = 0.01$) et que le coût total des événements de surchauffe est de 1 ($CO = 1$). Nous avons donc :

$$CS = 1 - \frac{(2 + 11) \times 0.01 + (1 - 0.66) \times 1}{1} = 0.53$$

Dans ce cas, notre solution serait en mesure de réduire le coût des événements de surchauffe pour le système de 53%. Le calcul du coût des événements de surchauffe CO est détaillé dans le prochain paragraphe.

4.4.2 Calcul du coût des événements de surchauffe

Pour calculer CO, nous pouvons utiliser la même méthode que pour calculer CA. Nous considérons NO le nombre d'intervalles décrit par les entrées dans les journaux systèmes associées à une surchauffe et C_o le coût moyen d'un événement de surchauffe par intervalle. Nous avons donc $CO = C_o \times NO$.

Pour être capable de calculer C_o, nous devons connaître les conséquences d'un événement de surchauffe sur le système. Nous avons montré dans le chapitre 3 que la conséquence

directe est la baisse de fréquence du processeur. Cette baisse de fréquence induit une augmentation de la durée d'exécution de l'application. L'augmentation de la durée d'exécution dépend à la fois de la baisse de fréquence (notée FO), de la durée de cette baisse (notée TO) mais également du type d'application considéré. Nous expliquons les hypothèses réalisées pour déterminer ces 3 facteurs pouvant influer sur l'augmentation de la durée d'exécution.

Influence du type d'application considéré

Nous avons mis en évidence dans le paragraphe 3.5.4 que pour les applications limitées par le processeur, une réduction de fréquence de 1% induit une augmentation du temps d'exécution de 1%. Pour les applications limitées par d'autres composants, l'augmentation du temps d'exécution est variable. Pour la suite de cette étude, nous considérons que les événements de surchauffe n'arrivent que lorsqu'une application est limitée par le processeur. En effet, nous pensons que le processeur doit être utilisé à son maximum pour arriver à un état de surchauffe. Par conséquent, nous considérons dans cette évaluation qu'une réduction de fréquence de 10% durant 1 minute augmentera la durée de l'application de $10\% \times 1\ minute = 6\ secondes$.

Baisse de fréquence

Pour l'estimation de FO, la documentation d'Intel donne de nombreuses informations à propos des différents paliers de fréquence accessibles en utilisant le mécanisme DVFS. Malheureusement, les données de supervision utilisées ne contiennent pas d'information permettant de connaître précisément la baisse fréquence des processeurs durant un événement de surchauffe. Dans la suite, nous considérerons donc cette baisse de fréquence comme variable et comprise entre une baisse de 30% et de 100% de la fréquence du processeur.

Durée de la baisse de fréquence

Pour pouvoir évaluer la durée d'un événement de surchauffe, nous utilisons les informations fournies par les entrées correspondantes à un événement de surchauffe dans les journaux systèmes du supercalculateur de DKRZ.

Cependant, nous ne disposons pas de données permettant de calculer la durée exacte pendant laquelle un processeur baisse sa fréquence lors des événements de surchauffe. Par conséquent, nous allons encadrer cette durée par une borne inférieure et une borne supérieure. Pour calculer la borne supérieure, nous allons simplement utiliser le nombre d'entrées présentes dans les journaux qui décrivent des événements de surchauffe. Pour calculer la borne inférieure, nous allons utiliser le nombre d'interruptions levées pour reporter la surchauffe d'un processeur.

Le calcul de la borne supérieure utilise donc le nombre d'entrées présentes dans les journaux systèmes. Pour rappel à chaque fois que la température d'un processeur dépasse le seuil maximum fixé par le constructeur, le processeur génère une interruption. Cette interruption est capturée par le noyau Linux. Le choix est fait dans le noyau Linux de ne générer qu'une entrée toutes les 5 minutes dans les journaux systèmes [4] et non pas à chaque interruption levée par le processeur. Lorsque la première interruption est levée, le noyau écrit la première entrée. Puis, si dans les 5 minutes suivant la première entrée il y a eu au moins une nouvelle interruption, alors une nouvelle entrée est écrite. Nous pouvons donc

4. https ://elixir.bootlin.com/linux/latest/source/arch/x86/kernel/cpu/mcheck/therm_throt.c

utiliser la durée entre deux entrées décrivant une surchauffe dans les journaux systèmes pour déterminer notre première borne. En effet, nous savons qu'une nouvelle entrée n'est écrite que s'il y a eu au moins un dépassement du seuil de température depuis la dernière entrée. Nous savons aussi que la durée entre ces deux entrées est fixée à 5 minutes. Par conséquent, si la prochaine entrée décrivant une surchauffe survient plus de 5 minutes après la dernière entrée, alors c'est un nouvel événement de surchauffe et le précédent événement s'est terminé avec la dernière entrée. Cependant, cette information n'est pas précise car nous ne savons pas précisément combien de temps le processeur est resté dans un état de surchauffe entre les entrées. Nous pouvons cependant considérer qu'au maximum le processeur est resté dans un état de surchauffe durant 5 minutes (la durée entre deux entrées). Ce temps nous fournit notre borne supérieure.

Pour le calcul de la borne inférieure, nous utilisons le nombre total d'interruptions qui est décrit par le champ *total events* du message de l'entrée correspondante. Ce nombre est remis à zéro à chaque redémarrage du nœud. Chaque cœur du processeur incrémente le compteur et il n'y a qu'un seul compteur pour l'ensemble du processeur. Pour connaître le nombre d'interruptions depuis la dernière entrée indiquant une surchauffe, il faut soustraire la valeur actuelle du compteur à la valeur précédente en ajoutant un pour prendre en compte l'interruption décrite par la première entrée et en divisant par le nombre de cœurs du processeur. Pour estimer la durée de l'événement de surchauffe, nous devons donc estimer la durée de la baisse de fréquence associée à une interruption. Cependant, la durée liée à une interruption n'est pas fixe. En effet, le processeur reste dans un état de surchauffe tant que sa température ne descend pas sous le seuil de température fixé par le constructeur. Une nouvelle interruption ne sera levée que si la température du processeur passe sous le seuil de température puis remonte au-dessus. De plus, pour éviter d'osciller rapidement entre fréquence nominale et fréquence réduite lorsque la température du processeur est proche de 0, de l'hystérésis est ajoutée pour que la baisse de fréquence soit effective durant un minimum de temps. Cependant, la durée de l'hystérésis n'est pas fixe, elle est calculée par le processeur à chaque nouvelle interruption [5]. Malheureusement, nous n'avons pas accès dans nos données à la durée précise de l'hystérésis ajoutée à chaque interruption.

Nous avons par conséquent deux méthodes permettant d'estimer la durée d'un événement de surchauffe. Cependant, aucune de ces méthodes n'est assez précise pour déterminer précisément cette durée. Nous allons donc considérer un encadrement de cette durée par une borne inférieure donnée par la durée minimale d'une interruption multipliée par le nombre total d'interruptions observées et une borne supérieure calculée en considérant qu'un processeur surchauffe durant toute la durée entre deux entrées appartement au même événement de surchauffe, soit 5 minutes.

À l'aide des informations de notre jeu de données, nous pouvons calculer les différentes valeurs nécessaires pour le calcul des deux bornes. Pour simplifier nos calculs dans le cas de la borne inférieure, nous considérons le nombre moyen d'interruptions observées entre deux entrées au lieu de considérer pour chaque entrée le nombre exact d'interruptions observées. Cette simplification n'influence que très marginalement les résultats de nos calculs.

Le nombre d'intervalles entre deux entrées correspondant à un événement de surchauffe est de $NA = 266$. Le nombre maximum d'interruptions entre deux entrées est de 9 090 et la durée entre deux entrées est de 5 minutes. La durée minimale de la baisse de fréquence associée à une interruption est donc de 33 ms (5 minutes divisées par 9 090). Le nombre

5. https ://www.intel.com/content/dam/www/public/us/en/documents/datas heets/xeon-e5-1600-2600-vol-1-datasheet.pdf

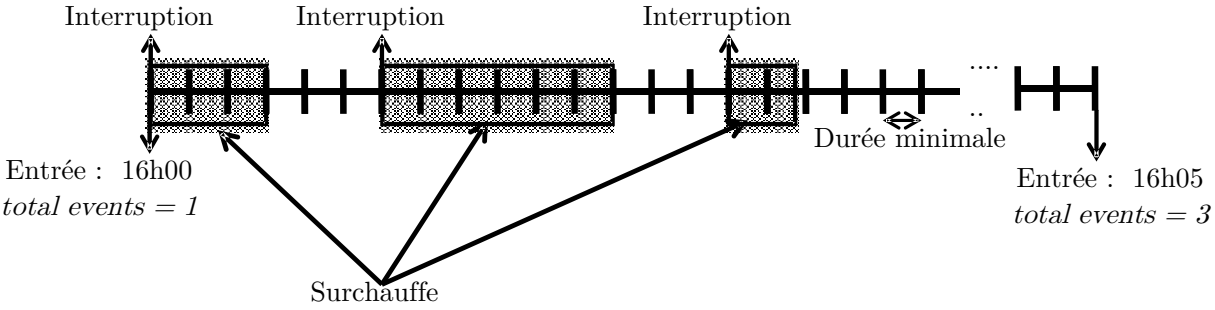


FIGURE 4.6 – Relation entre les interruptions et la durée de la surchauffe.

total d'interruptions est de 88 042. Le nombre moyen d'interruptions entre deux entrées est donc de 330 interruptions (88 042 divisé par 266).

Nous pouvons maintenant déterminer ces deux bornes à partir des valeurs précédentes. Le temps minimal de baisse de fréquence entre deux entrées TO_{min} est de $33ms \times 330 = 10,9$ *secondes* et le temps maximal de baisse de fréquence entre deux entrées TO_{max} est de 5 minutes.

Pour mieux comprendre le calcul des bornes minimales et maximales, la figure 4.6 présente la relation entre la durée de surchauffe, la durée des interruptions et la durée entre deux entrées.

Sur cet exemple, nous représentons 3 interruptions arrivant entre deux entrées (la première entrée est à 16h00, la deuxième entrée est à 16h05). Le compteur *total events* est à zéro au début. Nous considérons un processeur à un cœur pour plus de simplicité. Lors de la première interruption, le compteur est incrémenté de 1. Lors de la dernière interruption, la valeur du compteur est de 3 ce qui nous indique qu'il y a eu 3 interruptions. La durée entre les deux entrées est de 5 minutes. Notre borne minimale, calculée à partir du nombre total d'interruptions, vaudra donc $3 \times 33ms = 99ms$. Notre borne maximale, calculée à partir du temps entre deux entrées, vaudra simplement $1 \times 5minutes = 5minutes$. La durée réelle de la surchauffe sur cet exemple est de $11 \times 33ms = 363ms$.

Bien que les deux bornes nous permettent d'encadrer la durée totale de la surchauffe, nous voyons qu'elles ne sont pas précises. Pour la suite de l'évaluation, nous considérons donc que le temps réel de la surchauffe est compris entre nos deux bornes sans pouvoir donner d'estimation plus précise.

Nous avons défini le coût des surchauffes dans notre système à partir des données dont nous disposons. Nous pouvons maintenant évaluer si les capacités prédictives de la solution sont suffisantes pour permettre aux actions préventives prises de réduire le coût des surchauffes.

4.4.3 Réduction de fréquence anticipée

La première action proposée est une réduction de fréquence anticipée. Pour éviter les événements de surchauffe, nous proposons d'appliquer une faible réduction de fréquence sur le processeur quand un tel événement est prédit. Pour expliquer pourquoi nous pouvons attendre une réduction du coût des événements de surchauffe en utilisant une action de ce type, nous devons étudier l'influence d'une réduction de fréquence du processeur à la fois sur la température du processeur et sur la performance de l'application.

L'influence de la fréquence du processeur sur la durée d'exécution de l'application est déterminée dans le paragraphe 3.5.4. Nous considérerons dans la suite de cette évaluation

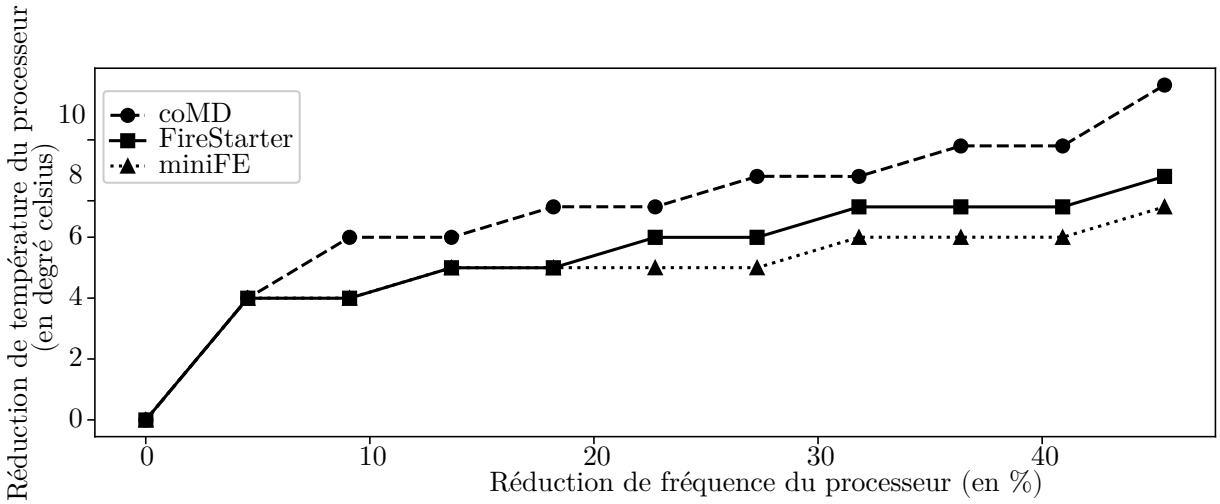


FIGURE 4.7 – Variation de la température en fonction de la fréquence du processeur.

le pire cas, c'est-à-dire qu'une baisse de fréquence de 1% entraîne une augmentation du temps d'exécution de l'application de 1%.

Influence d'une réduction de fréquence sur la température du processeur

Pour évaluer l'influence d'une réduction de fréquence sur la température du processeur, nous avons utilisé un système hébergé en interne. En effet, à cause des permissions requises sur le système pour fixer la fréquence du processeur, il n'est pas possible d'utiliser directement le système hébergé par DKRZ. Cependant, le système hébergé en interne est similaire à celui utilisé chez DKRZ. Il est refroidi par eau tiède et dispose des processeurs de même génération (2 processeurs Intel E5-2698 et 48 Go de mémoire vive par nœud).

La méthodologie pour nos expériences est la suivante. Nous démarrons et laissons fonctionner une application durant 15 minutes à la fréquence maximale du processeur pour laisser la température du processeur augmenter et se stabiliser. Puis, nous appliquons une baisse de fréquence d'une minute sur un processeur et nous observons la baisse de température de ce processeur. Nous considérons 3 applications pour nos expériences : deux applications représentatives des applications utilisées dans un système HPC (coMD et miniFE) fournies par le NERSC et FireStarter [6] qui est un programme spécifiquement construit pour atteindre le maximum de dissipation thermique d'un processeur. Les résultats présentés sont la moyenne de 3 tests pour chaque application et chaque baisse de fréquence.

La figure 4.7 montre l'influence de la réduction de fréquence du processeur sur sa température pour les 3 applications. La première remarque est que la variation de la température du processeur n'est pas linéaire. Pour toutes les applications, une baisse de la fréquence de 4% implique une baisse significative de la température de 4°C. Après cette première importante baisse, la température continue de décroitre mais plus lentement. La présence de plateaux sur les courbes est due à la faible précision de la DTS (1°C). En complément, nous pouvons noter que la baisse de température du processeur intervient très rapidement après la baisse de fréquence (moins de 2 secondes dans nos expériences).

En utilisant ces résultats, nous décidons de supposer qu'une réduction de fréquence de 4% appliquée 1 minute avant un événement de surchauffe et durant 1 minute est capable

6. https ://github.com/tud-zih-energy/FIRESTARTER/ commit cbab04f

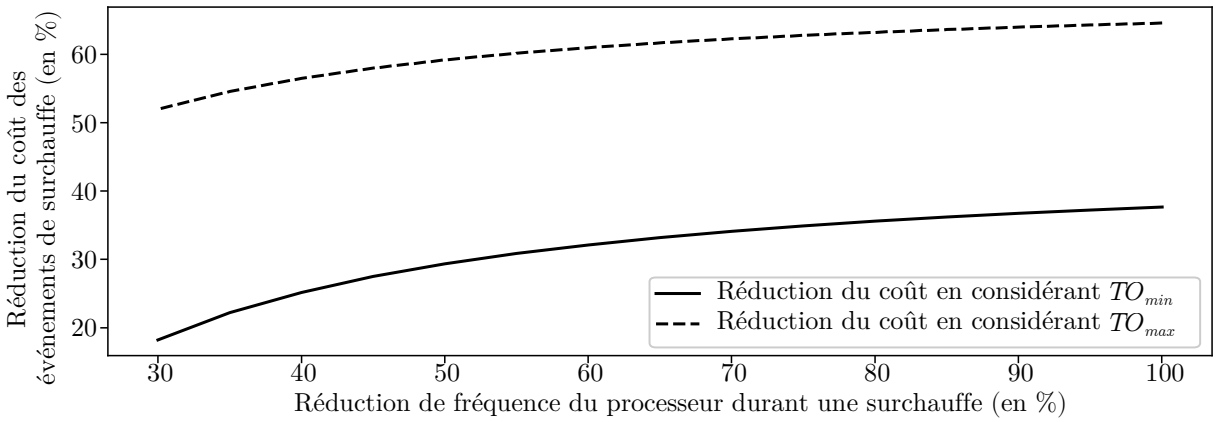


FIGURE 4.8 – Variation de la réduction du coût des surchauffes de notre solution en considérant la baisse de fréquence anticipée en fonction de la fréquence des processeurs durant un événement de surchauffe.

d'éviter l'occurrence de cet événement. Il est à noter que nous considérons que la baisse de fréquence doit être *obligatoirement* appliquée 1 minute avant l'événement. En d'autres termes, une baisse de fréquence appliquée 2 minutes avant ne sera pas capable d'éviter l'événement de surchauffe. C'est une supposition conservative car nous pouvons facilement envisager qu'une baisse de fréquence appliquée quelques minutes en avance est capable d'éviter l'événement de surchauffe en permettant un refroidissement du processeur.

Coût de l'action préventive

Pour calculer le coût de l'action préventive, nous devons prendre en compte la baisse de fréquence correspondante à l'action (notée FA dans la suite et égale à 4%), la durée de cette baisse de fréquence ($TA = 1\ minute$) et le nombre total d'actions (NA) qui est le nombre total de prédictions réalisées par notre solution. Par conséquent, le coût de l'action préventive est donné par $CA = NA \times TA \times FA$. Dans cette évaluation, nous considérons qu'à chaque prédiction de surchauffe réalisée par notre solution, la fréquence du processeur sera baissée de 4% pendant 1 minute.

La figure 4.8 montre la variation de la réduction du coût des surchauffes par la baisse de fréquence anticipée utilisant les prédictions réalisées par notre solution en fonction de la fréquence des processeurs durant un événement de surchauffe. L'évaluation est réalisée en considérant TO_{min} et TO_{max}. Pour chaque évaluation, nous sélectionnons les paramètres de notre solution prédictive en optimisant la réduction du coût pour TO_{min} et pour TO_{max}. De plus, l'optimisation de notre solution est réalisée en considérant une réduction de la fréquence du processeur de 70% lors d'une surchauffe c'est-à-dire que nous ne ré-entraînons par notre modèle pour optimiser ces résultats sur chaque réduction de fréquence possible. L'optimisation de la solution en considérant la baisse de fréquence préventive revient à maximiser la valeur de $Ri]0, 1]$ tout en minimisant le nombre de prédictions NA.

Nous notons CS_{max} la réduction du coût en considérant TO_{max} et CS_{min} la réduction du coût en considérant TO_{min}. Pour optimiser la valeur de CS_{min}, nous obtenons $Ri]0, 1] = 0.21$ et $NA = 103$. Pour optimiser la valeur de CS_{max}, nous obtenons $Ri]0, 1] = 0.63$ et $NA = 1434$. Les paramètres sélectionnés sont résumés dans le tableau 4.3.

La figure 4.8 montre que, même en utilisant des suppositions conservatives à propos de l'action préventive, la réduction du coût des événements de surchauffe est toujours supérieure à 0. De plus, la réduction du coût maximale est de 35% en considérant TO_{min} et

			Considérant TO_{min}		Considérant TO_{max}	
Action	C_a	I	NA	CS_{min}	NA	CS_{max}
Baisse de fréquence anticipée	2.4s	]0,1]	103	28%	1434	58%
Migration de tâche	30s	[1,40]	18	-6%	312	46%

Tableau 4.6 – Résumé des paramètres et des résultats en utilisant une action préventive en considérant une baisse de fréquence du processeur de 50% durant un événement de surchauffe ($FO = 50\%$). I est l'intervalle de temps avant un événement de surchauffe durant lequel une action préventive est considérée comme capable de l'éviter.

63% en considérant TO_{max}. À cause des différents mécanismes impliqués dans la réduction de fréquence du processeur lorsqu'un événement de surchauffe, il est difficile de savoir quelle est la valeur la plus réaliste. En supposant une baisse de fréquence de 50% en moyenne durant les événements de surchauffe, la réduction du coût réalisée par la combinaison de l'action préventive et de notre solution prédictive est comprise entre 28% et 58%. Les résultats sont résumés dans le tableau 4.6.

Nous avons aussi calculé la réduction du coût attendue lorsque la solution prédictive repose sur l'utilisation d'un seuil. Dans ce cas, la réduction de coût maximale en considérant une baisse de fréquence de 50% lors d'une surchauffe serait de 50%. En considérant TO_{min}, la solution proposée reposant sur un seuil n'est pas capable de réduire le coût des surchauffes. En effet, elle augmente le coût de celles-ci de 192% à cause d'un nombre important de Faux Positifs entraînant des baisses de fréquences anticipée trop fréquentes. En utilisant le seuil avec un oracle capable de prédire les 5 prochaines valeurs de température, nous obtenons $CS_{max} = 55\%$ et $CS_{min} = -98\%$ (augmentation du coût de 98%). Ces résultats montrent clairement le bénéfice d'utiliser notre solution prédictive par rapport à une solution basique utilisant un seuil.

Nous allons maintenant évaluer la deuxième action envisagée.

4.4.4 Migration de tâches

Une autre action envisagée peut être la migration de tâches. L'idée est de déplacer la tâche de calcul en cours sur le processeur qui va générer une surchauffe sur un autre processeur qui est *froid*. La définition de tâche est ici générique : il s'agit de l'ensemble des processus applicatifs s'exécutant sur un processeur.

Plusieurs travaux estiment le coût d'une migration de tâche. Wang et al. [93] estiment le coût d'une migration en utilisant les applications provenant de la suite de tests de performance NAS [7]. Ils exécutent les applications sur 16 nœuds distincts et utilisent une migration au niveau du processus pour estimer le surcoût de la migration. Ils concluent que le surcoût d'une migration de tâche est compris entre 1 et 8 secondes. Nagarajan et al. [68] utilisent les mêmes applications mais utilisent cette fois-ci une migration en utilisant l'hyperviseur Xen au lieu d'une migration au niveau du processus. Ils tirent les mêmes conclusions et trouvent un surcoût compris entre 1 et 16 secondes pour la majorité des applications.

En s'appuyant sur ces travaux, nous considérons un surcoût de 30 secondes par migration. C'est une supposition conservative car nous prenons en compte que les résultats fournis par les auteurs précédant utilisent des applications ayant une empreinte en mémoire vive restreinte. En considérant notre cas d'usage, le système HPC de DKRZ dispose d'une

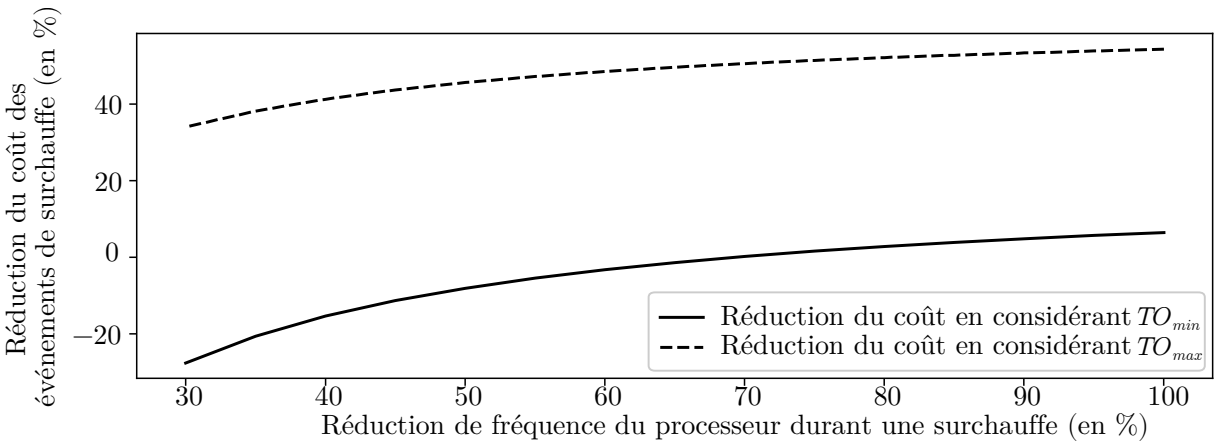


FIGURE 4.9 – Variation de la réduction du coût des surchauffes de notre solution en considérant la migration de tâches en fonction de la fréquence des processeurs durant un événement de surchauffe.

bande passante entre les nœuds de 6 Go/s et les nœuds ont une capacité de mémoire vive maximale de 64 Go, il faudrait donc 11 secondes pour transférer entièrement la mémoire vive d'un nœud vers un autre nœud de calcul. En tenant compte du surcoût de la migration elle-même (ralentissement des tâches de calcul, sauvegarde de la mémoire avant transfert, etc.), nous pensons que considérer un surcoût complet de 30 secondes est raisonnable. Par conséquent, le coût de cette action est $CA = NA \times 30$ *secondes*.

Pour que la migration de tâches soit utile et soit capable d'éviter les événements de surchauffe, nous considérons que l'action doit être prise au minimum 1 minute avant l'événement de surchauffe (ceci prend en compte le temps nécessaire à la migration de 30 secondes) et au plus 40 minutes avant un événement de surchauffe. De plus, si l'action est prise dans cet intervalle, nous considérons que notre solution ne prédira plus d'autres événements de surchauffe car la tâche incriminée aura été déplacée sur un processeur *froid*. Autrement dit, nous considérons uniquement la première prédiction dans l'intervalle [1,40] et nous ignorons les autres. Cependant, l'ensemble des prédictions en-dehors de cet intervalle sont prises en considération pour calculer la valeur de NA.

La figure 4.9 présente la variation de la réduction du coût des surchauffes de notre solution en considérant la migration de tâches en fonction de la fréquence des processeurs durant un événement de surchauffe. L'évaluation est réalisée pour TO_{min} et TO_{max}. De la même manière que précédemment, pour chaque évaluation nous sélectionnons la combinaison des paramètres optimisant la réduction du coût pour TO_{min} et TO_{max}. En considérant l'action de migration de tâches, l'optimisation revient à maximiser $Ri[1, 40]$ tout en minimisant NA. En optimisant CS_{min}, nous obtenons $Ri[1, 40] = 0.46$ et $NA = 18$. En optimisant CS_{max}, nous obtenons $Ri[1, 40] = 0.7$ et $NA = 312$. Ces résultats sont résumés dans le tableau 4.6. Nous voyons que par rapport à l'action de baisse de fréquence, le nombre de prédictions NA de la solution est ici beaucoup plus faible (14 par rapport à 108 en considérant TO_{min} et 312 par rapport à 1434 en considérant TO_{max}). Cela s'explique par le coût de l'action. La baisse de fréquence est moins coûteuse, il est donc possible de faire plus de prédictions pour réussir à prédire plus d'événements de surchauffe.

La figure 4.9 montre que la réduction maximale du coût des événements de surchauffe est de 4% en considérant TO_{min} et de 50% en considérant TO_{max}. En comparant ces résultats avec la réduction de fréquence anticipée, nous pouvons noter que la réduction du coût des événements de surchauffe est plus faible en considérant la migration de tâches. En

effet, la migration de tâches est plus coûteuse que la réduction de fréquence anticipée. Par conséquent, en considérant TO_{min}, le coût des événements de surchauffe est trop faible par rapport au coût de la migration pour être utile si la fréquence du processeur lorsqu'une surchauffe est inférieure à 70%. En considérant TO_{max}, la conclusion change car le coût des événements de surchauffe devient plus important que le coût de la migration de tâches et par conséquent la réduction des coûts est toujours supérieure à 0.

De la même manière que précédemment, nous pouvons calculer la réduction des coûts des événements de surchauffe pour les méthodes utilisant un seuil. Dans ce cas, à cause du coût plus élevé de l'action, les méthodes utilisant un seuil échouent à réduire le coût de la surchauffe. En considérant la réduction des coûts maximale et TO_{min}, la méthode utilisant un seuil entraîne une augmentation du coût de 3400% et en considérant TO_{max} l'augmentation est de 98%. En utilisant un seuil avec un oracle capable de prédire les 5 prochaines valeurs de température, l'augmentation est de 980% en considérant TO_{min} et elle est de 15% en considérant TO_{max}.

4.4.5 Synthèse

À travers cette évaluation incluant des actions préventives, nous avons vu que les résultats fournis par notre solution sont suffisants pour permettre de réduire le coût de la surchauffe dans notre système. Les deux actions préventives évaluées montrent aussi que notre solution peut s'adapter aux différentes caractéristiques des actions pour optimiser au mieux la réduction du coût. Dans le cas de la migration de tâche, il est préférable de prédire longtemps en avance mais de réduire au maximum les faux positifs à cause du coût de l'action. Pour la réduction anticipée de fréquence, il est préférable de maximiser les prédictions 1 minute avant l'événement de surchauffe tout en minimisant les faux positifs.

Nous allons maintenant présenter une discussion sur les résultats présentés et sur la méthode proposée.

4.5 Discussion

Dans cette section, nous discutons de trois points importants liés à notre solution. Le premier point est l'influence de la modification de température du processeur par les actions prises sur les prédictions réalisées par la solution. Le deuxième point est le choix des actions préventives. Le troisième et dernier point est l'utilisation d'une approche de prédiction des températures en complément de notre solution d'apprentissage machine.

Les actions préventives prises pour éviter un événement de surchauffe peuvent influer sur les futures prédictions de la solution. En effet, ces actions modifient la température du processeur et par conséquent modifient la forme de la courbe de la température. Notre solution repose sur l'analyse de cette forme pour être capable de prédire les prochains événements de surchauffe. Il est donc important de savoir si les changements de la forme de la courbe vont avoir une influence sur les capacités prédictives de notre solution, et si cette influence est positive (en évitant des faux positifs) ou négative (manquer des événements de surchauffe). Notre solution utilise de l'apprentissage machine pour être capable de prédire les événements de surchauffe à partir de données de petite taille (moins de 100 exemples) incluant du bruit et capturée avec un faible taux d'échantillonnage. Bien que nous soyons confiants dans la capacité de notre solution à apprendre des nouveaux motifs liés à l'activation des actions préventives menant à un événement de surchauffe, il serait nécessaire d'évaluer dans un contexte réel l'influence sur nos résultats. Cette évaluation

n'a pu être réalisée car elle requiert d'utiliser un système en production avec notre solution pour être capable de récupérer des données de température incluant l'influence des actions envisagées.

Les actions préventives proposées dans ce chapitre sont des actions simples à mettre en œuvre qui influent uniquement sur le processeur en surchauffe. D'autres actions plus complexes peuvent être envisagées. Ces actions peuvent utiliser des approches différentes ou améliorer les approches actuellement étudiées. Par exemple, dans notre cas d'étude de la migration de tâches, nous étudions la migration sur n'importe quel nœud du système. Or, il apparaît évident que la migration sur un nœud physiquement proche est plus optimale et permettrait de réduire le surcoût de celle-ci en perturbant le moins possible le système et l'application. Nous pourrions aussi imaginer une augmentation temporaire de la puissance de la pompe de refroidissement dans le cas d'un refroidissement liquide pour accroître la capacité de dissipation thermique et donc éviter la surchauffe. Cette augmentation temporaire ne pourrait être appliquée que durant un court instant car elle influerait négativement sur l'efficacité énergétique du système. Une autre approche pourrait être d'utiliser le gestionnaire de ressources pour éviter d'allouer les nœuds proches et pour permettre de réduire légèrement la température autour du processeur sur lequel un événement de surchauffe est prédit. Enfin, dans notre cas d'étude, deux processeurs sont refroidis en série par le même flux d'eau. Une action pourrait utiliser cette propriété pour baisser la fréquence des deux processeurs refroidis à l'aide du même flux pour optimiser le refroidissement obtenu à l'aide de la baisse de fréquence.

Nous avons évalué la borne supérieure des résultats que pouvait donner une méthode qui utilise une approche fondée sur la prédiction des prochaines valeurs de la température en utilisant un seuil et oracle. Cependant, cette approche de prédiction des prochaines valeurs de la température peut aussi être utilisée dans le cadre d'une méthode utilisant les événements précurseurs pour prédire une surchauffe pour améliorer ces résultats et augmenter le temps d'avance des prédictions. En effet, notre solution peut être adaptée en fonction de l'optimisation recherchée. En utilisant un module de prédiction des températures, nous pourrions adapter de nouveau notre solution pour optimiser les résultats en tenant compte de celui-ci et cela pourrait permettre de détecter les événements précurseurs avec une avance plus grande. Cependant, l'évaluation des solutions de prédiction des températures est un travail à part entière. En effet, il faut dans un premier temps évaluer quelle méthode propose les meilleurs résultats et il en existe un grand nombre : régression logistique, modèle ARIMA, RNN, etc. De plus, la construction elle-même de la méthode doit être évaluée : devons-nous prendre en entrée toutes les températures ou juste les dernières minutes ? Devons-nous avoir un modèle par nœud, par armoire ou un seul modèle pour l'ensemble du système ? Ce point est une perspective intéressante pour des travaux futurs sur le sujet.

4.6 Synthèse

Dans ce chapitre, nous avons proposé une solution utilisant un algorithme d'apprentissage machine pour prédire les événements de surchauffe et nous avons évalué ses résultats en calculant la réduction du coût des surchauffes attendue en utilisant des actions préventives.

Nous proposons une solution qui analyse la température des processeurs pour prédire les prochains événements de surchauffe. Elle est découpée en deux grandes étapes : la mesure de similarité entre les séries temporelles représentant l'évolution de la température des processeurs et l'apprentissage des corrélations entre l'évolution de la température et

les événements de surchauffe à l'aide d'un algorithme d'apprentissage machine.

Pour réaliser la mesure de similarités, nous utilisons un algorithme proposé par l'état de l'art : la *Dynamic Time Warping* (DTW). Cet algorithme mesure la distance entre deux courbes de manière *optimale* ce qui permet de pouvoir caractériser des données peu précises grâce à l'analyse de leur similarité. Cette similarité des courbes caractérisée par la DTW est ensuite utilisée pour établir les corrélations entre les données de température et les surchauffes.

Ces corrélations sont apprises par un algorithme d'apprentissage machine. Le choix de l'algorithme d'apprentissage machine a été réalisé selon les caractéristiques des données disponibles pour l'apprentissage. Notre jeu de données ne dispose que de peu d'exemples de surchauffe (moins de 100 dans notre cas). L'algorithme d'apprentissage machine choisi à partir de ces caractéristiques est le *Gradient Boosting Tree*. Grâce à ces capacités importantes de généralisation, il est capable d'apprendre des corrélations en utilisant des jeux de données contenant très peu d'exemples.

Pour optimiser les résultats produits par le *Gradient Boosting Tree*, un module d'agrégation est utilisé à la fin de l'algorithme d'apprentissage. Ce module analyse la sortie fournie par l'algorithme d'apprentissage machine en utilisant des corrélations temporelles pour améliorer la précision de la solution.

L'analyse des résultats de l'ensemble de la chaîne de traitement montre que la solution proposée est capable de prédire 76% des événements de surchauffe 5 minutes en avance avec une précision de 76%.

Nous avons réalisé une comparaison de notre approche utilisant la DTW et le *Gradient Boosting Tree* avec deux méthodes utilisant un seuil : une première méthode utilisant un seuil seul sur la température du processeur et une deuxième méthode utilisant un seuil sur la température prédite avec 5 minutes d'avance du processeur grâce à un oracle. La comparaison des résultats permet de conclure que les meilleurs résultats sont atteints en utilisant l'algorithme du *Gradient Boosting Tree*.

Finalement, nous évaluons la pertinence globale de la solution en estimant sa capacité à réduire le coût des surchauffes dans notre système. Cette évaluation est réalisée en utilisant deux actions préventives qui utilisent les prédictions faites par notre solution. La première action préventive consiste à réduire faiblement la fréquence du processeur après une prédiction de surchauffe. La deuxième préventive consiste à migrer les tâches du processeur sur lequel la prédiction est réalisée vers un autre processeur *froid*. L'évaluation réalisée montre qu'en utilisant des suppositions conservatives à la fois sur le coût des surchauffes et sur le coût des actions, la réduction du coût des surchauffes sur notre système peut aller jusqu'à 63% grâce à la combinaison de notre solution prédictive et des actions préventives proposées.

Pour finir, le calcul du surcoût de notre solution permet de conclure qu'il est négligeable pour le système.

Nous pouvons conclure que la solution développée dans ce chapitre est capable de prédire les surchauffes des processeurs en tenant compte des caractéristiques des données d'entrées (faible précision, peu d'exemples, etc.) tout en fournissant des résultats assez fiables pour permettre une réduction jusqu'à 63% des coûts des surchauffes dans le système.

Chapitre 5

Analyse des journaux systèmes

Les systèmes HPC produisent des grandes quantités de données de supervision. Nous nous intéressons dans ce chapitre à l'analyse des journaux systèmes. Ces journaux contiennent des informations textuelles sur les événements se produisant dans le système. Ils peuvent décrire des événements différents comme le démarrage d'une application, l'augmentation de la température d'un système ou encore la connexion d'un utilisateur. L'ensemble de ces événements est une source d'information importante pour décrire l'état du système et aider à la compréhension des défaillances.

Cependant l'analyse de ces journaux n'est pas facile. En effet, l'ensemble des composants et des applications fonctionnant sur un système HPC écrivent des entrées dans les journaux systèmes. Ceci génère une grande quantité de données à traiter. De plus, la diversité des événements décrits par ces entrées est également importante.

Dans ce chapitre, nous proposons une solution permettant d'analyser automatiquement ces journaux pour mettre en évidence les relations causales entre les différentes entrées composant ces journaux pour aider à l'analyse de ceux-ci. Nous avons nommé cette solution LogFlow. Cette solution est constituée de deux parties. La première partie est l'identification et le regroupement des entrées présentes dans les journaux systèmes qui décrivent le même événement. La deuxième partie est la mise en évidence des relations causales entre les événements détectés par l'analyse automatique de ceux-ci. Il est à noter que nous utiliserons le nom LogFlow par la suite à la fois pour parler de la partie attachée au regroupement des entrées et pour parler de la partie responsable de la mise en évidence des relations causes.

L'approche globale choisie s'articule donc autour de deux étapes distinctes. La première étape est de regrouper les entrées présentes dans les journaux systèmes qui décrivent le même événement. Une fois que cette étape est réalisée, nous obtenons une suite d'événements qui se déroulent dans le système HPC. L'étape suivante est de détecter les relations causales entre ces événements. Pour réaliser cette détection, nous utilisons un modèle prédictif de manière détournée. Le but de ce modèle est de prédire l'événement actuel à l'aide des événements précédents. Si la prédiction est correcte, alors nous savons qu'il existe des relations qui ont permises de prédire l'événement actuel à partir des événements précédents. La dernière étape consiste à déterminer les relations qui ont permis au modèle de prédire l'événement actuel. Cette mise en évidence des relations nécessite d'analyser le modèle prédictif pour comprendre quelles informations ont été utilisées pour réaliser la prédiction.

Le début de ce chapitre est consacré à l'exposé des problématiques liées à la construction de la solution et à la définition du vocabulaire utilisé par la suite. La section 5.4 décrit l'analyseur de journaux que nous proposons pour regrouper les entrées décrivant un

événement similaire et présente les résultats de celui-ci sur plusieurs jeux de données de l'état de l'art. La section 5.5 décrit la solution proposée pour détecter les relations causales entre les entrées des journaux systèmes et elle présente les résultats de cette solution sur le jeu de données de DKRZ. Enfin, la section 5.6 discute des limitations de la solution et des possibles solutions. La section 5.7 conclut les travaux présentés dans ce chapitre.

5.1 Vocabulaire

Nous commençons par décrire le vocabulaire utilisé dans la suite de ce chapitre.

Les mots *journaux systèmes* décrivent l'ensemble des journaux où les services systèmes écrivent des informations à propos de leur état. Chaque ligne de ces journaux est appelée une *entrée*. Chaque *entrée* est composée de plusieurs champs dont le champ *message*. Le champ *message* représente la partie non structurée des entrées décrivant les *événements*.

Nous considérons qu'une *entrée* est écrite quand un *événement* se produit. A chaque fois qu'un événement *équivalent* se produit, une entrée contenant un message *similaire* est générée. Nous décrivons ici des entrées *similaires* comme des entrées qui partagent le même motif d'écriture. Ce motif sera décomposé en une partie fixe égale pour toutes les entrées *similaires* et en une partie variable qui peut être spécifique à chaque entrée. La partie fixe de ce motif sera appelée un *schéma*.

5.2 Problématique

Les journaux systèmes sont écrits par l'ensemble des composants du système HPC et des applications fonctionnant sur celui-ci. Ces journaux systèmes peuvent décrire le cycle de vie d'une application (son démarrage, ses étapes de fonctionnement et son arrêt, etc.) des événements survenant dans le système (changement de température ou connexion d'un utilisateur, etc.), mais également rapporter des informations fournies par les applications et logiciels sur leur exécution (perte d'une connexion vers le système de fichiers, délai d'attente dépassé pour l'envoi d'une donnée ou encore non disponibilité d'une interface réseau). Par conséquent, les entrées qui composent ces journaux présentent des corrélations entre elles : la connexion d'un utilisateur sera suivie de sa déconnexion, la perte de connexion vers le système de fichiers entraînera une défaillance l'application, etc. La solution proposée dans ce chapitre permet de mettre automatiquement en évidence ces corrélations. Nous allons maintenant détailler les différents défis inhérents à la construction d'une solution automatique de mise en évidence des relations causales dans les journaux systèmes. Puis nous détaillerons l'approche globale utilisée pour la mise en œuvre de la solution.

5.2.1 Les défis

L'élaboration d'une solution permettant d'analyser et de mettre en évidence les corrélations présentes entre les entrées des journaux systèmes nécessite de relever différents défis. Nous décrivons ici les principaux défis liés à l'élaboration de la solution.

Identification des entrées représentant un événement équivalent

L'un des premiers défis est d'être capable d'identifier les entrées qui correspondent à un événement équivalent dans le système. En effet, un même événement peut être décrit

par deux entrées qui ne sont pas strictement identiques. L'exemple suivant expose deux entrées présentes dans les journaux systèmes associées au même événement :

```
a) Connection of user A54890
b) Connection of user B89010
```

Bien que les entrées décrivent le même événement (la connexion d'un utilisateur ici), elles ne sont pas strictement identiques (le nom d'utilisateur est ajouté à la fin de l'entrée). Il est donc nécessaire de pouvoir lier les deux entrées au même événement pour être ensuite capable de détecter les corrélations entre l'événement de connexion d'un utilisateur et d'autres événements survenant dans le système.

Détection des relations causales entre les événements décrits par les entrées des journaux systèmes

Une fois que les événements sont correctement associés aux entrées dans les journaux systèmes, nous souhaitons pouvoir leur donner plus de valeur que le simple ordre chronologique dans lequel ils sont écrits dans les journaux systèmes. En effet, l'ordre chronologique peut ne pas être suffisant pour mettre en évidence les événements qui sont corrélés dans les journaux. Ceci est dû au fait que la corrélation initiale qui peut être décrite par les entrées n'apparaît pas forcément de manière évidente car différents services systèmes peuvent écrire en même temps dans le même journal système et les entrées peuvent être entrelacées.

La figure 5.1 montre un journal système décrivant une défaillance survenant sur une application. Cette défaillance est décrite par le message "Défaillance application, impossible d'ouvrir le fichier". Elle décrit un problème lors de l'accès d'un fichier par une application ce qui entraîne la défaillance de l'application. Cette défaillance peut être expliquée par les entrées présentes en gras dans la figure. Ainsi, la défaillance de l'application est due à une erreur d'accès sur un fichier "Erreur accès fichier" et "Accès au fichier impossible". Cette erreur est elle-même due à une défaillance sur le réseau "Perte de connexion vers Lustre". Enfin, les messages "Accès au fichier demandé" et "Connexion aux nœuds de stockage Lustre" sont aussi corrélés car ils décrivent le début du flux qui mène à cette défaillance.

Nous voyons ici la mise en évidence de ces corrélations n'est pas évidente. Elles sont entrecoupées par des lignes provenant d'autres services systèmes et il est *à priori* compliqué de définir l'ensemble des messages liés à la défaillance sans une analyse manuelle du journal.

Journal système

10h00 Réseau	**Connexion aux noeuds de stockage Lustre**
10h01 Service identification	Connexion d'un utilisateur
10h06 Service identification	Déconnexion d'un utilisateur
10h10 Surveillance disque	Augmentation de la température du disque
10h11 Système de fichiers	**Accès au fichier demandé**
10h15 Service identification	Connexion d'un utilisateur
10h20 Surveillance processeur	Augmentation de la température du processeur
10h22 Réseau	**Perte de connexion vers Lustre**
10h24 Service identification	Déconnexion d'un utilisateur
10h25 **Système de fichiers**	**Accès au fichier impossible**
10h25 **Système de fichiers**	**Erreur accès fichier**
10h27 Surveillance disque	Diminution de la température du disque
10h30 Surveillance application	**Défaillance application, impossible d'ouvrir le fichier**
11h00 Service identification	Connexion d'un utilisateur

FIGURE 5.1 – Exemple d'entrelacement des entrées des services systèmes dans les journaux systèmes.

Il est donc nécessaire de mettre au point une méthode d'analyse automatique de ces journaux qui soit capable de détecter les relations causales entre les événements malgré l'entrelacement des informations fournies par les entrées.

Exploitation des données disponibles et choix techniques

Les systèmes HPC produisent une grande quantité de données de supervision. Dans notre cas, le système de DKRZ produit plus d'un milliard d'entrées dans les journaux systèmes par an (section 3.1). Cette volumétrie de données représente un défi et il est nécessaire de la prendre en compte lors de la construction de la solution. En effet, il faut à la fois choisir des méthodes capables d'analyser de très grande quantité de données mais également garder un temps de traitement et un surcoût pour le système *raisonnable*. Si le surcoût pour le système est trop important (de par la durée de traitement et la capacité de calcul nécessaire), l'utilité de la solution sera discutable car il sera compliqué de pouvoir augmenter la disponibilité du système en consommant des ressources importantes pour la solution elle-même. Ce défi est à prendre en compte à la fois pour la partie regroupant les entrées pour la découverte des événements mais également pour la partie mettant en évidence les relations causales entre les événements. Comme nous le verrons dans la section 5.3.1, les méthodes existantes permettant d'associer les entrées et les événements ne permettent pas d'analyser un volume important de données en un temps *raisonnable*.

Enfin, la solution doit pouvoir être utilisée simplement par un administrateur ou un utilisateur du système. Pour que cette utilisation soit le plus simple possible, elle doit être entièrement automatisée. Cela inclut donc l'automatisation des deux parties distinctes de la solution : la détection des événements dans les journaux systèmes et la mise en évidence des relations causales entre les événements dans les journaux systèmes. Dans la première partie, comme nous le verrons dans la section 5.3.1, il existe de nombreuses solutions pour détecter les événements à partir des entrées dans les journaux systèmes. Cependant, ces solutions nécessitent, pour la plupart, un paramétrage manuel qui peut se révéler complexe pour un utilisateur. Ce paramétrage manuel consiste à optimiser des paramètres internes de la solution utilisée pour obtenir la meilleure détection possible. Dans la seconde partie, nous utilisons un algorithme d'apprentissage machine. De la même manière, ces algorithmes peuvent présenter un nombre important de paramètres à fixer pour optimiser les résultats (taux d'apprentissage, choix du nombre et du type de couches pour les réseaux de neurones, etc.).

Nous souhaitons donc que la solution développée ici ne nécessite aucun paramétrage manuel car celui-ci peut être complexe pour un utilisateur et les résultats de la solution peuvent entièrement dépendre de celui-ci.

En conclusion, nous avons besoin d'une méthode permettant d'analyser un grand volume de données complexes composées des journaux systèmes pour mettre en évidence les corrélations des événements décrits par ces journaux systèmes. La solution doit aussi rester simple pour être utilisable rapidement par un administrateur ou un utilisateur du système.

5.3 Etat de l'art

L'état de l'art présenté ici se découpe en deux parties. La première traite du regroupement des entrées avec les événements correspondants. La deuxième se focalise sur l'utilisation de méthodes d'apprentissage profond pour l'analyse des entrées des journaux systèmes.

5.3.1 Regroupement des entrées décrivant un événement similaire

Nous allons dans cette partie décrire les travaux proposant des solutions pour regrouper les entrées qui décrivent un événement similaire. Nous commençons par définir le format des journaux système et par expliciter les pré-requis nécessaires à la détection des événements. Puis nous détaillons les travaux de l'état de l'art.

Format des journaux systèmes

La structure d'une entrée dans un journal système suit la norme RFC 5424 [1]. La structure définie plusieurs champs : la sévérité (*severity*), la catégorie (*facility*), l'horodatage (*timestamp*), le nom de la machine (*hostname*), le nom de l'application (*app-name*) et le message. Chaque champs décrit une propriété de l'entrée et peut être composé de valeur prédéfinies ou non. Le tableau 5.2 décrit ces champs et leurs valeurs prédéfinies. Il est à noter qu'il existe d'autres types de journaux que les journaux systèmes. Nous pouvons citer les journaux applicatifs par exemple. Dans ce cas, bien que la plupart du temps les champs soient identiques, les applications peuvent ajouter des champs supplémentaires selon leur besoin.

Bien que tous les champs ne disposent pas de valeurs prédéfinies, le nombre de valeurs possibles pour un champ est en général faible ou les valeurs des champs sont facilement analysables automatiquement. Ainsi, le champ `"catégorie"` contient la catégorie de l'équipement du système HPC ayant généré l'entrée et ne comporte que 7 valeurs possibles dans notre cas, le champ `"horodatage"` peut très facilement être analysé automatiquement car le motif est fixe (dans le cas de l'exemple le motif est "$annee - mois - jourTheure : minute : seconde$"), le champ `"nom de la machine"` contient le nom du nœud de calcul et est donc aussi facilement analysable car le motif sera fixe (composé d'une lettre et d'un numéro dans notre cas), et le `"nom de l'application"` ne comporte que 74 valeurs différentes possibles dans notre cas. En effet, nous étudions ici uniquement les entrées des journaux systèmes. Le champ `"nom de l'application"` ne décrit donc que l'ensemble des services systèmes fonctionnant sur le système HPC. Les entrées sont écrites de manière chronologique et un seul fichier est écrit par nœud de calcul et par mois.

Cependant l'ensemble de ces champs ne suffit pas à décrire un événement. En effet, il faut aussi prendre en compte le champ `"message"` qui est plus complexe à analyser. Nous allons voir dans le prochain paragraphe deux entrées partageant des champs communs mais décrivant des événements différents.

Détection d'un événement

Le tableau 5.1 présente deux exemples avec des valeurs de champs identiques sauf pour le champ message. Nous pouvons voir que même si les deux entrées sont reliées à des événements liés et partagent des valeurs de champs identiques (sauf pour `"message"`),

1. https ://tools.ietf.org/html/rfc5424

Horodatage	Nom de la machine	Catégorie	Sévérité	Nom de l'application	Message
2018-02-01T09 :40 :31	m20226	authpriv	info	sshd	session opened for user root by (uid=0)
2018-02-01T09 :40 :33	m20226	authpriv	info	sshd	Received disconnect from 168.265.23.963 : disconnected by user

Tableau 5.1 – Exemple de deux entrées dans les journaux systèmes.

elles ne décrivent pas le même événement : la première décrit la connexion d'un utilisateur alors que la deuxième décrit la déconnexion d'un utilisateur.

Nous devons donc nous intéresser plus particulièrement à la valeur du champ `"message"` pour pouvoir attribuer précisément un événement à une entrée. Comme nous pouvons le voir dans les exemples donnés dans le tableau 5.1, le champ `"message"` ne suit pas une norme précise. Il est composé de parties fixes qui décrivent l'événement et de parties variables qui décrivent les données associées à l'événement. Dans la première entrée, le nom de l'utilisateur "root" et la valeur de son uid "(uid=0)" sont des parties variables, le reste étant une partie fixe. Dans le deuxième exemple, la partie variable est l'adresse de l'utilisateur "168.265.23.963" et la partie fixe est le reste du message.

Nous allons maintenant décrire les travaux existants de l'état de l'art qui présentent des solutions pour regrouper les entrées différentes décrivant des événements *équivalents*.

Présentation des travaux de l'état de l'art

Le regroupement des entrées décrivant des événements *équivalents* dans les journaux système est largement traité dans l'état de l'art. Ce regroupement est communément appelé analyseur de journaux (*logs parser*). L'évaluation de la qualité d'un analyseur repose sur deux caractéristiques : sa rapidité et sa précision. En effet, bien qu'il soit évident qu'une détection précise soit requise, il doit être aussi capable de traiter en un temps *raisonnable* l'ensemble des entrées produites par le système.

Zhu et al. [108] proposent un ensemble de tests afin d'évaluer 13 méthodes différentes sur 16 jeux de données provenant de différents systèmes. Les 16 ensembles de données sont décrits dans le tableau 5.3. Ils sont découpés en 3 grands types : les données provenant d'une application, les données provenant d'un système HPC et les données provenant d'un système d'exploitation. La dernière ligne correspond à nos données. Ces données sont de taille variable : de 21329 entrées pour Proxifier à plus de 211 millions d'entrées pour Thunderbird. Notre jeu de données provenant de DKRZ est le plus gros jeu de données, il contient plus de 660 millions d'entrées.

Zhu et al. proposent également un résumé et une catégorisation des méthodes qui sont reportées dans le tableau 5.4. La catégorisation est composée de 7 stratégies utilisées par les analyseurs de journaux : découverte des motifs fréquents (*frequent pattern mining*), groupage (*clustering*), partitionnement itératif (*iterative partitioning*), plus grande sous-séquence commune (*longest common subsequence*), analyseur d'arbre (*parsing tree*), algorithme génétique (*evolutionary algorithms*) et autre stratégie. Les méthodes utilisant

Champs	Contenu	Valeurs prédéfinies	Exemple
Sévérité	Sévérité de l'entrée	De plus grave au moins grave : urgence (*emergency*), alerte (*alert*), critique (*critical*), erreur (*error*), avertissement (*warning*), notice (*notice*), informationnel (*informational*), débogage (*debug*)	avertissement
Catégorie	Catégorie du service système ayant généré l'entrée	/	Réseau
Horodatage	Heure d'écriture de l'entrée. La précision dépend du système.	/	2019-09-10T23 :20 :50
Nom de la machine	Nom de la machine sur lequel le système écrivant l'entrée fonctionne	/	marc-PC
Nom de l'application	Nom de l'application écrivant l'entrée	/	firefox
Message	Message décrivant l'événement	/	Démarrage de l'application

Tableau 5.2 – Définition des champs contenus dans les entrées des journaux systèmes. Le caractère "/" signifie qu'il n'y a pas de valeur prédéfinie.

Nom du jeu de données	Description	Type	Nombre d'entrées	Poids
HDFS	Journaux du système de fichiers HDFS	Applicatif	11175629	1.47 Go
Hadoop	Journaux du cadriciel de calculs distribués Hadoop	Applicatif	394 308	48.61 Mo
Spark	Journaux du cadriciel de calculs distribués Spark	Applicatif	33236604	2.75 Go
ZooKeeper	Journaux du logiciel ZooKeeper	Applicatif	74380	9.95 Mo
OpenStack	Journaux du logiciel OpenStack	Applicatif	207820	58.61 Mo
HealthApp	Journaux du logiciel HealthApp	Applicatif	253395	22.44 Mo
Apache	Journaux du serveur Apache	Applicatif	56481	4.9 Mo
OpenSSH	Journaux du serveur OpenSSH	Applicatif	655146	70.02 Mo
Proxifier	Journaux du logiciel Proxifier	Applicatif	21329	2.42 Mo
BGL	Journaux systèmes du système HPC Blue Gene/L	HPC	4747963	708.76 Mo
HPC	Journaux systèmes d'un système HPC	HPC	433489	32 Mo
Thunderbird	Journaux systèmes du système HPC Thunderbird	HPC	211212192	29.60 Go
Windows	Journaux systèmes du sous-système CBS (*Component Based Servicing*) de Windows provenant d'un ensemble d'ordinateurs	Système d'exploitation	114608388	26.09 Go
Linux	Journaux systèmes de Linux	Système d'exploitation	25567	2.25 Mo
Mac	Journaux systèmes de Mac OS	Système d'exploitation	117283	16.09 Mo
Android	Journaux du système d'exploitation Android	Système d'exploitation	1555005	183.37 Mo
DKRZ	Journaux systèmes du système HPC DKRZ	HPC	662781666	72 Go

Tableau 5.3 – Données utilisées pour évaluer les analyseurs de journaux.

la découverte de motifs fréquents reposent sur la même heuristique : parcourir une ou plusieurs fois le jeu de données en construisant à chaque parcours un ensemble d'items fréquents. Les entrées sont ensuite décrites par ces ensembles d'items fréquents et elles sont regroupées par entrées partageant les mêmes ensembles. Les méthodes utilisant sur le grou-

Méthode	Catégorie	Rapidité	Précision moyenne	Nombre de paramètres
SLCT [87]	Découverte des motifs fréquents	+++	0.63	1
AEL [50]	Autre heuristique	+++	0.77	2
IPLoM [60]	Partitionnement itératif	+++	0.80	2
LKE [36]	Groupage	+	0.65	1
LFA [67]	Découverte des motifs fréquents	+++	0.64	0
LogSig [84]	Groupage	++	0.56	1
SHISO [65]	Groupage	+++	0.68	4
LogCluster [88]	Découverte des motifs fréquents	+++	0.68	1
LenMa [80]	Groupage	++	0.77	1
LogMine [41]	Groupage	++	0.74	3
Spell [27]	Plus grande sous-séquence commune	+++	0.79	1
Drain [45]	Analyseur d'arbres	+++	0.87	0
MoLFI [63]	Algorithme évolutionnaire	+	0.62	0

Tableau 5.4 – Présentation des analyseurs de journaux et de leurs propriétés.

page estiment des distances entre les entrées. En utilisant sur les distances calculées, elles vont regrouper ensemble les entrées ayant des distances proches. Les méthodes utilisant le partitionnement itératif découpent successivement les entrées en groupes de mots. Les méthodes utilisant la plus grande sous-séquence commune groupe les entrées qui partagent la même plus grande sous-séquence commune. Les méthodes utilisant un analyseur d'arbre construisent une représentation des entrées se fondant sur des arbres. Chaque nœud de l'arbre représente une règle pour catégoriser les entrées. Une feuille de l'arbre représente un ensemble d'entrées correspondant au même événement. Les algorithmes génétiques sont utilisés lorsque le regroupement des entrées en événements est vu comme un problème à optimisations multiples qui est à la fois de maximiser le nombre d'entrées regroupées ensemble et de maximiser la spécificité de chaque événement. Ce problème peut être résolu en utilisant des algorithmes génétiques.

Zhu et al. présentent une évaluation des méthodes selon deux critères principaux : la rapidité d'exécution des méthodes et la précision de ces méthodes.

La rapidité des méthodes utilisées par ces auteurs est une notion vague non précisée par les auteurs. Nous mesurerons par la suite de manière précise le temps d'exécution de l'ensemble des méthodes sur une partie des jeux de données présentés dans le tableau 5.3.

Les auteurs évaluent ensuite la précision de chaque méthode sur chaque jeu de données.

Pour être capable de noter la précision, ils utilisent un sous-ensemble annoté de 2000 entrées pour chaque jeu de données. Ce sous-ensemble est annoté à la main par leurs soins. La précision calculée est une précision stricte c'est à dire que l'événement correspondant à une entrée doit être strictement égal à l'événement annoté pour cette entrée.

Pour relier les événements détectés et les événements annotés par les auteurs, un numéro unique est attribué à chaque événement différent. La relation est ensuite réalisée en associant à chaque événement annoté l'événement détecté qui apparait le plus souvent lors cet événement annoté.

Dans le cas où deux événements sont détectés pour un seul événement annoté par une méthode, les entrées appartenant à l'événement détecté avec la plus haute fréquence d'apparition pour l'événement annoté sont considérées comme justes et les entrées correspondant à l'autre événement sont considérées comme fausses. La précision moyenne est calculée comme la moyenne des précisions sur l'ensemble des 16 jeux de données. Les détails peuvent être trouvés sur la documentation proposée par les auteurs [2]. Il est à noter que nous n'avons pas été capables de retrouver l'ensemble des précisions fournis par les auteurs sur les plus gros jeux de données car nous n'avons pas été capables de faire fonctionner les méthodes proposées sur l'entièreté des jeux de données (Thunderbird notamment). De la même façon, nous n'avons pas été capables de retrouver les temps d'exécution sur certaines méthodes fournis par He et al. [44]. Une discussion avec les auteurs est en cours pour comprendre les possibles erreurs d'utilisation de leur outil [3]. Il est à noter cependant qu'en considérant les temps d'exécution reportés par He et Al. [44], la conclusion à propos du temps d'exécution du paragraphe 5.4.4 reste inchangée.

Le tableau 5.4 permet de voir dans un premier temps qu'il n'y a pas de catégorie de méthodes supérieure à d'autres en tenant compte de la rapidité d'exécution et de la précision moyenne. De plus, les méthodes peuvent être plus ou moins compliquées à configurer à cause d'un nombre important de paramètres qui nécessité d'être fixé avant l'exécution. Par exemple, la méthode SHISO nécessite de fixer 4 paramètres spécifiques avant son exécution. La détermination des valeurs de ces paramètres demande une phase de test avant l'utilisation de la méthode pour trouver les meilleurs paramètres possibles. A l'inverse certaines méthodes comme LFA ou Drain ne demandent pas de paramètres à fixer. Il n'est pas nécessaire de passer par une phase de test pour obtenir les meilleurs résultats possibles.

Ce tableau nous permet aussi de conclure qu'actuellement Drain représente le meilleur analyseur de l'état de l'art avec une précision moyenne de 0.87. Cependant, la rapidité d'exécution donnée par les auteurs étant une notion vague, nous ne pouvons pas conclure sur la méthode la plus rapide, ni sur celle présentant le meilleur compromis entre rapidité d'exécution et précision moyenne. Dans la suite de nos travaux, nous nous comparons donc par rapport à Drain mais aussi par rapport aux autres méthodes pour évaluer de manière plus précise la rapidité de chaque méthode.

5.3.2 Méthodes d'apprentissage profond pour l'analyse des événements des journaux systèmes

Après avoir identifié les événements présents dans les journaux systèmes, nous pouvons utiliser des méthodes d'apprentissage profond pour détecter les relations entre ces événements. Nous allons dans un premier temps expliquer le fonctionnement des méthodes

2. https ://logparser.readthedocs.io/en/latest/benchmark.html
3. https ://github.com/logpai/logparser/issues/46

d'apprentissage profond utilisées par la suite. Puis nous détaillerons les travaux utilisant ces méthodes pour analyser les journaux systèmes.

Description des méthodes d'apprentissage profond

Une introduction au fonctionnement des méthodes d'apprentissage profond est réalisée dans la section 2.3.3. Nous nous focalisons donc ici sur le fonctionnement des réseaux de neurones récurrents qui est un type de méthode d'apprentissage profond car c'est ce type de réseau que nous utiliserons par la suite. Nous détaillons ensuite le fonctionnement d'un type de réseau récurrent : le LSTM.

Nous utilisons des réseaux de neurones récurrents car ce sont des réseaux spécialisés dans l'analyse des séries temporelles et l'analyse des journaux systèmes est similaire à l'analyse des séries temporelles. Chaque nouvelle entrée dans les journaux correspond à un point de données ordonné chronologiquement. Ainsi, nous pouvons utiliser les algorithmes d'apprentissage profond spécialisés dans l'analyse des séries temporelles : les réseaux de neurones récurrents (RNN).

Le principe des réseaux de neurones récurrents est de pouvoir traiter des signaux temporels et de pouvoir échanger des informations à l'intérieur d'une même couche de neurones pour aider à la compréhension de la temporalité du signal. Ainsi, le réseau peut utiliser des informations du passé (des entrées 1 et 2) pour mieux *comprendre* l'entrée 3 comme le montre la figure 5.2.

Cependant, un des principaux problèmes du réseau de neurones récurrents est qu'il est difficile pour une information présente au début de la couche (par exemple une information présente sur le premier neurone) d'influer sur un neurone présent à la fin de la même couche (par exemple le centième neurone de la couche). La figure 5.3 présente en détails une couche d'un réseau de neurones récurrents. E représente l'entrée du réseau, P représente les différents poids et S les sorties du réseau. De manière plus formelle, l'état interne H d'un neurone t est donné par :

$$H_t = f(H_{t-1}, E_t) \; avec \; f \; la \; fonction \; d'activation \tag{5.1}$$

Si l'on considère la tangente hyperbolique ($tanh$) en tant que fonction d'activation nous avons :

$$H_t = tanh(P_{H_{t-1}} \times H_{t-1} + E_t \times P_{E_t}) \tag{5.2}$$

La sortie d'un neurone est donnée par :

$$S_t = P_{S_t} \times H_t \tag{5.3}$$

Nous voyons à travers ces équations qu'une information contenue dans le premier neurone va très rapidement s'effacer en passant de neurone en neurone dans la même couche. En effet, à chaque neurone traversé, elle sera multipliée par un poids, puis additionnée avec l'entrée du neurone elle aussi affectée d'un poids.

Pour résoudre ce problème, deux nouveaux types de réseaux de neurones récurrents ont été créés : les Long-Short Term Memory [48] (LSTM) et les Gated Recurrent Unit [18] (GRU). Ces deux types de réseau proposent une solution pour qu'une information soit capable de traverser la couche entière si elle est considérée comme importante. Bien que les deux types de réseaux utilisent une approche légèrement différente, ils ont des performances comparables. Cependant, les GRU ont théoriquement moins de paramètres et sont donc capable d'apprendre plus vite. Dans la pratique, la vitesse d'apprentissage et

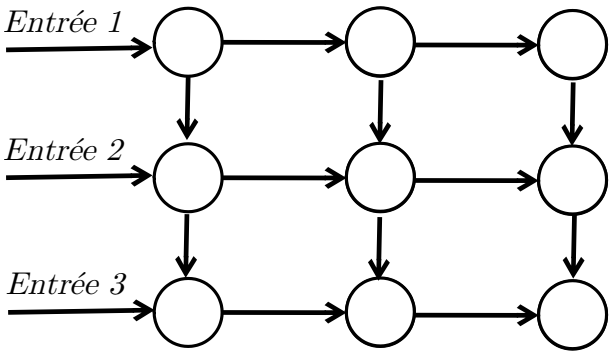


FIGURE 5.2 – Réseau de neurones récurrents.

les résultats obtenus dépendent du problème et du jeu de données. Dans notre cas d'étude, le test des deux types de réseau n'a pas montré de différences significatives sur la vitesse d'apprentissage ou sur les résultats. Nous avons donc choisi d'utiliser les LSTM pour leur implémentation dans un large choix de cadriciels dédiés aux réseaux de neurones. Nous nous focalisons par conséquent uniquement la présentation de ceux-ci.

La figure 5.4 présente de manière détaillée un neurone de type LSTM. Nous considérons pour cette figure qu'il s'agit du deuxième neurone d'une couche. Il utilise ainsi comme entrée la sortie du neurone 1 (représentée par C_1 et H_1) et la deuxième valeur du vecteur d'entrée E_2. Ses sorties C_2 et H_2 sont connectées au troisième neurone de la couche. Pour plus de simplicité, les poids ne sont pas représentés. Le neurone est composé de trois grandes parties : la partie A est appelée porte d'oubli, la partie B est appelée porte d'entrée et la partie C est appelée porte de sortie.

Détaillons chacune de ces parties. La partie A est la partie qui détermine l'importance de l'état avant C_1 dans la sortie de la cellule C_2. La partie B reprend l'idée du réseau récurrent en utilisant l'état interne de la cellule précédente H_1 et l'entrée courante E_2 pour moduler l'information courante en fonction du passé. Enfin la partie C est la partie "sortie". Elle utilise à la fois la valeur calculée par la porte d'oubli A additionnée à la porte d'entrée B, à la valeur de l'entrée E_2 et à la valeur de l'état interne de la cellule précédente H_1. Cette construction permet soit d'ignorer ce qui vient de l'état précédent soit de ne garder que l'état précédent grâce à la porte d'oubli (l'état précédent peut aussi être modulé plus finement).

Les fonctions f_1, f_2, f_4 sont le plus souvent des fonctions sigmoid (abréviées σ) alors que les fonctions f_3, f_5 sont classiquement des tangentes hyperboliques. f_2^A, f_2^B et f_2^C représentent respectivement la sortie de la porte d'oubli A, la sortie de la porte d'entrée B et la sortie de la porte de sortie C.

De manière formelle, l'équation de la partie A est :

$$f_t^A = \sigma_1(P_{E_t}^1 \times E_t + P_{H_{t-1}}^1 \times H_{t-1}) \tag{5.4}$$

L'équation de la partie B est donnée par :

$$f_t^B = f_t^A \times C_{t-1} + \sigma_2(P_{E_t}^2 \times E_t + P_{H_{t-1}}^2 \times H_{t-1}) \times tanh_3(P_{E_t}^3 \times E_t + P_{H_{t-1}}^3 \times H_{t-1}) \tag{5.5}$$

L'équation de la partie C est donnée par :

$$f_t^C = \sigma_4(P_{E_t}^4 \times E_t + P_{H_{t-1}}^4 \times H_{t-1}) \times tanh_5(f_t^B) \tag{5.6}$$

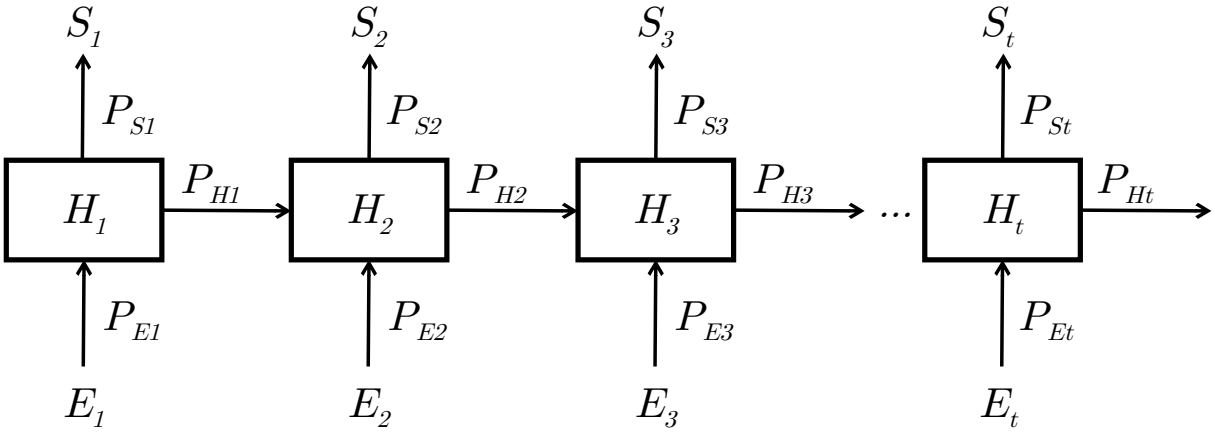


FIGURE 5.3 – Présentation détaillée d'un réseau de neurones récurrents.

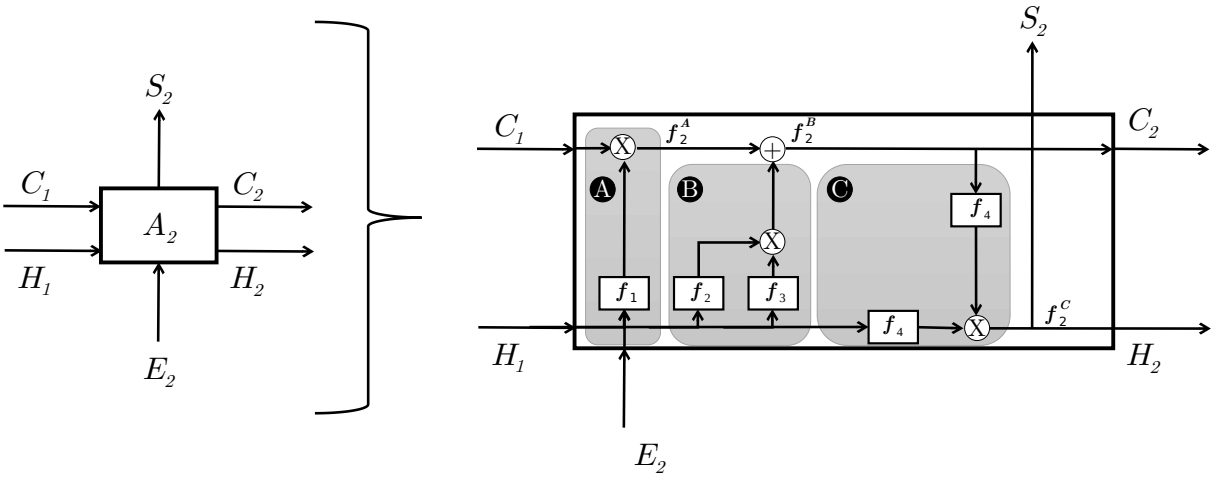


FIGURE 5.4 – Présentation détaillée d'un neurone de type LSTM.

avec :

P_x^i le poids associé à la fonction i appliquée sur le vecteur x (5.7)

σ_i la fonction d'activation de la fonction i (5.8)

$\times$ le produit matriciel de Hadamard (5.9)

t le numéro du neurone (5.10)

Grâce à ces trois portes (entrée, sortie et oubli), ce type de neurone peut apprendre des récurrences lointaine si elles sont importantes dans la prédiction.

Interprétation des informations apprises des réseaux de neurones

Un des principaux désavantages des réseaux profonds est le fonctionnement en *boite noire*. L'optimisation des poids du réseau pour une prédiction rend difficile l'interprétation des informations utilisées pour réaliser cette prédiction car le réseau construit une représentation interne de l'information. Cette représentation interne est très difficilement interprétable car elle relève de l'optimisation de fonctions mathématiques et elle fait appel à l'optimisation de plusieurs milliers, voire millions, de poids. Il est donc difficile de *savoir* quelles informations le réseau a utilisées pour réaliser sa prédiction. Pour notre utilisation du réseau de neurones, l'interprétabilité est importante. En effet, pour rappel,

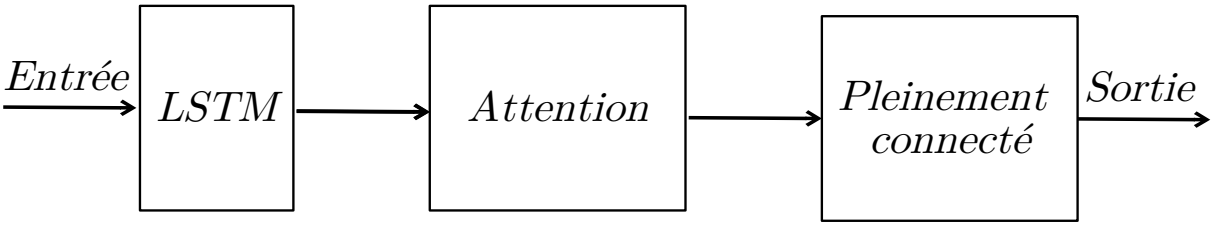


FIGURE 5.5 – Exemple d'un LSTM utilisant une couche d'attention.

nous souhaitons demander au réseau de prédire l'événement actuel et si la prédiction est correcte nous utiliserons le réseau pour en déduire les relations entre l'événement prédit et les événements précédents. La déduction de ces corrélations nécessite de pouvoir interpréter les informations apprises et utilisées par le réseau pour réaliser sa prédiction.

Pour résoudre ce problème *d'interprétabilité*, des nouvelles couches ont fait leur apparition : les couches d'attention. Leur but est de permettre d'interpréter les résultats du réseau et de savoir quel est l'élément de l'entrée que le réseau utilise pour prédire la sortie.

De nombreux types de couches d'attention existent selon le type de réseau utilisé [42, 59, 89, 100]. Nous nous focaliserons ici sur les couches d'attention dédiées aux LSTM [59, 72, 90].

La figure 5.5 présente l'ajout de la couche d'attention pour permettre d'interpréter le réseau. Comme précédemment expliqué, un réseau d'apprentissage profond est généralement composé de plusieurs couches. Ici, nous représentons un réseau composé de deux couches de base : une couche avec des neurones de type LSTM et une couche de type pleinement connecté. La couche d'attention vient se placer entre la couche LSTM et la couche pleinement connectée. Celle-ci va interagir avec le LSTM pour calculer des poids. Chaque poids indique l'importance de l'entrée dans la prédiction. Ces poids sont ensuite utilisés par la couche pleinement connectée en plus de la sortie du LSTM. La figure 5.6 montre un exemple de l'utilisation de la couche d'attention.

Dans cet exemple, nous utilisons le réseau pour prédire le sentiment lié à la phrase d'entrée. Dans cet exemple, la phrase *"Je suis heureux d'être ici"* est donnée en entrée du réseau. Chaque mot correspond à un neurone d'entrée du LSTM. Cette phrase est utilisée en entrée du LSTM. Le LSTM fournit ensuite une représentation interne des informations (qui correspond au premier vecteur de nombres sur la figure). Une valeur haute ici n'est pas forcément associée à un poids important dans la prédiction. Ce vecteur est ensuite utilisé par la couche d'attention qui nous fournit en sortie l'importance de chaque entrée. La représentation interne du LSTM et les poids de la couche d'attention sont ensuite multiplés entre eux et fournis à une couche pleinement connectée. Enfin, la sortie du réseau est la probabilité de la phrase d'être négative ou positive. Dans l'exemple, le réseau donne une probabilité de 0.96 que la phrase soit positive et une probabilité de 0.04 que la phrase soit négative. Grâce à la couche d'attention, nous pouvons savoir que le réseau utilise principalement le mot *heureux* pour réaliser sa prédiction. C'est principalement ce mot qui lui permet de prédire le sentiment positif de la phrase d'entrée.

Nous avons considéré pour l'exemple qu'un LSTM pouvait utiliser des données de type textuel en entrée. Cependant comme la majorité des algorithmes d'apprentissage machine, il nécessite des données numériques en entrée. Nous détaillons dans la prochaine partie le prétraitement nécessaire pour utiliser un LSTM sur des données textuelles.

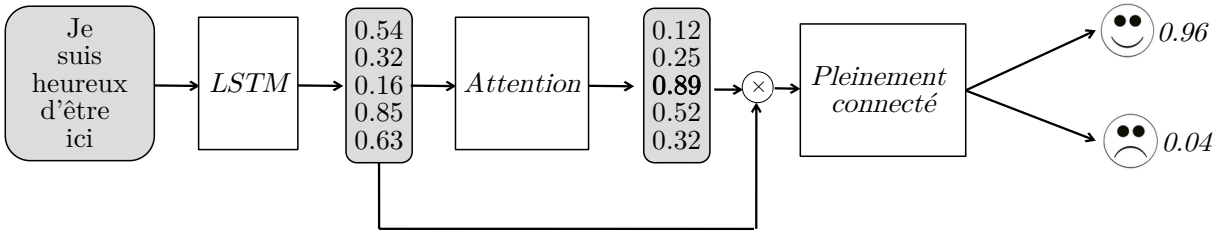


FIGURE 5.6 – Exemple d'un LSTM utilisant une couche d'attention pour la classification.

Pré-traitement des données d'entrée

Dans l'exemple de la figure 5.6, nous avons considéré que le LSTM utilisait en entrée directement les mots de la phrase pour simplifier l'explication du modèle. Les algorithmes d'apprentissage machine ne peuvent pas utiliser directement des données de type texte en entrée, elles doivent être transformées en valeur numérique.

Une première méthode est d'attribuer à chaque mot un identifiant unique, puis de transformer la phrase en un vecteur d'identifiants. Cependant cette méthode a un désavantage majeur : il est impossible de corréler la *distance* entre les mots et leurs significations. Ainsi si l'on considère les mots *"heureux"* et *"heureuse"*, il est logique de dire que ces mots sont proches sémantiquement et donc que leur distance doit être faible.

Pour ne pas devoir construire et ordonner un ensemble de mots à la main, plusieurs méthodes proposent de représenter les mots proches par des valeurs proches. Nous allons décrire la méthode la plus commune que nous utiliserons par la suite : *word2vec* [64]. Cette méthode permet de calculer une distance entre les mots et d'avoir par exemple, en considérant $d(x, y)$ la distance entre le mot x et le mot y, $d(\text{"}heureux\text{"}, \text{"}heureuse\text{"}) \approx 0$ alors que par exemple $d(\text{"}heureux\text{"}, \text{"}chien\text{"}) \gg 0$.

Cette méthode initialise chaque mot avec un vecteur de taille n aléatoire. Ensuite, en parcourant les données, les vecteurs des mots proches physiquement dans le texte sont rapprochés. Ainsi il est possible d'effectuer l'opération : $\text{"}roi\text{"} - \text{"}homme\text{"} + \text{"}femme\text{"}$. Cette opération nous donnera $\text{"}roi\text{"} - \text{"}homme\text{"} + \text{"}femme\text{"} = \text{"}reine\text{"}$.

Nous allons maintenant aborder l'utilisation des techniques d'apprentissage profond pour l'analyse des journaux systèmes.

Présentation des travaux de l'état de l'art

Des travaux récents utilisent des techniques d'apprentissage profond pour analyser les journaux systèmes à large échelle et plus spécifiquement pour détecter les anomalies dans les entrées des journaux systèmes [28, 13, 62]. Bien que notre solution diffère de ces travaux car notre but n'est pas de détecter les anomalies dans les entrées des journaux systèmes, nous pouvons utiliser les conclusions et idées des précédents travaux pour construire notre solution.

Les travaux présentés par Du et al. [28] sont parmi les premiers à présenter une solution utilisant des LSTM pour apprendre le comportement d'un système à large échelle en utilisant les entrées dans les journaux systèmes. Leur modèle est similaire au nôtre, le LSTM apprend à prédire le prochain événement dans les journaux en utilisant les événements précédents. Cependant, leur but est différent. La prédiction du LSTM sert à estimer si le prochain événement est un événement normal et attendu. En d'autres termes, si le prochain événement est différent de la prédiction réalisée par le LSTM, alors une potentielle anomalie est détectée. Il est à noter que la prédiction se réalise sur l'événement

mais aussi sur les métriques qu'il contient (code erreur, valeur de la température, etc.). Notre étude présente donc deux différences. La première est que nous nous intéressons à la détection des corrélations dans le but de présenter ces corrélations à l'utilisateur. Cette compréhension des corrélations détectées par le LSTM est très difficilement réalisable à cause du fonctionnement en *boîte noire*. Pour permettre cette compréhension nous avons utilisé une couche spécialisée dans l'analyse des corrélations : une couche d'attention. De plus, l'étude faite par les auteurs se limite à des jeux de données contenant quelques dizaines d'événements alors que notre étude s'intéresse à plusieurs centaines d'événements. Ceci implique une différence dans le traitement des données et, dans notre cas, l'utilisation de plusieurs modèles en parallèle.

Brown et al. [13] proposent d'utiliser une couche d'attention pour mieux comprendre les décisions prises par leur modèle utilisant un LSTM pour la détection des défaillances à partir des entrées dans les journaux systèmes. Ils se focalisent sur l'analyse de données de sécurité fournies par Los Alamos National Laboratory. Leur but est de permettre la détection des attaques réalisées par une équipe du laboratoire. Les attaques sont donc annotées dans le jeu de données mais ces annotations sont utilisées uniquement pour la notation de leur solution et non pour l'apprentissage. Leur solution permet de donner à chaque nouvelle entrée dans les journaux un score d'anomalie. Ce score est ensuite utilisé pour déterminer si l'entrée correspond à une attaque ou non. En utilisant leur idée d'introduire une couche d'attention, nos travaux montrent que les informations obtenues par les poids fournis par cette dernière peuvent être utilisés pour déduire les relations causales entre les entrées des journaux systèmes.

Meng et al. [62] étudient l'étape d'intégration des données. Leurs travaux s'intéressent à la meilleure représentation possible des entrées des journaux systèmes en tant que vecteur pour prendre en compte le sens des messages. A l'issue de leur étude, il propose une solution appelée template2vec qui est spécialement adaptée au cas de l'analyse des journaux systèmes. Notre solution utilise une approche classique (word2vec [64]) pour cette étape. Il serait intéressant de tester template2vec pour l'étape d'intégration et de mesurer l'amélioration, ou non, des résultats.

Enfin, Debnath et al. [22] étudient la parallélisation de l'analyse des entrées en utilisant le cadriciel Apache Spark. Ces travaux sont complémentaires aux nôtres. Cependant, en considérant le temps d'exécution de notre solution, tant en apprentissage qu'en test, en utilisant un seul nœud de calcul, nous n'avons pas considéré nécessaire d'étudier la parallélisation sur plusieurs nœuds.

5.3.3 Synthèse

Nous avons vu à travers l'étude de l'état de l'art qu'il existe des travaux connexes au nôtre.

En effet, il existe un nombre important de travaux traitant du regroupement des entrées présentes dans les journaux systèmes qui décrivent un événement *similaire*. Ces travaux proposent plusieurs approches différentes pour permettre ce regroupement. Lors de la présentation de notre solution, nous devrons donc nous comparer directement à ces travaux pour évaluer la pertinence de notre solution.

Bien qu'il existe aussi des travaux utilisant des méthodes d'apprentissage profond pour l'analyse des journaux systèmes, aucun de ceux-ci ne propose de les utiliser pour permettre de mettre en évidence les relations causales entre les événements présents dans les journaux systèmes. De plus, les jeux de données utilisés par ces travaux présentent des différences

soit au niveau de la volumétrie soit au niveau du type d'entrées analysées. Néanmoins ces travaux montrent qu'il est possible d'utiliser des réseaux de neurones récurrents pour analyser les journaux systèmes.

5.4 Regroupement des entrées des journaux décrivant un événement similaire

Nous allons maintenant présenter notre solution permettant le regroupement des entrées présentes dans les journaux systèmes décrivant un événement *similaire*.

L'objectif de ce regroupement peut être décrit de la façon suivante : nous cherchons à regrouper les entrées décrivant le même événement et à dissocier les entrées décrivant des événements différents. Résoudre ce problème revient à essayer de répondre à deux objectifs : à la fois de dissocier au maximum les entrées pour être précis sur l'événement détecté (le cas extrême étant d'avoir un événement différent pour chaque entrée) mais aussi à la fois d'essayer de regrouper au maximum les entrées qui décrivent un événement similaire pour garantir de regrouper *toutes* les entrées correspondant au même événement (le cas extrême étant d'avoir un événement correspondant à toutes les entrées).

Nous commençons par décrire l'approche utilisée pour réaliser ce regroupement, puis nous détaillons les contraintes à respecter par la solution. Nous détaillons ensuite notre approche qui utilise la découverte des parties fixes et variables dans les journaux systèmes. Enfin nous présentons les résultats de notre solution et les comparons par rapport aux méthodes de l'état de l'art.

5.4.1 Approche utilisée

Notre solution utilise la détection des parties fixes et variables des entrées pour regrouper efficacement les entrées décrivant le même événement. Cette approche est catégorisée comme "découverte des motifs fréquents" selon la classification proposée dans la section 5.3.1. Pour permettre une découverte efficace des motifs fréquents, nous faisons une hypothèse forte qui n'est pas réalisée dans les autres travaux. Nous supposons que les entrées qui partagent des motifs en commun doivent être de la même taille. Autrement dit, nous ne cherchons les motifs fréquents qu'entre des entrées ayant la même taille (*i.e.*, composées du même nombre de mots).

Cette hypothèse forte nous semble réaliste. En effet, nous pouvons considérer qu'un même événement sera toujours décrit par des entrées de même taille car les entrées sont décrites dans le code source des composants et des services systèmes qui les écrivent. Les entrées décrivant un même événement ne peuvent donc évoluer que si le code source du composant ou du service évolue. Or ce code source ne peut évoluer que lors d'une mise à jour des composants ou des services systèmes. Ces mises à jour des composants ou des services systèmes apportant une modification dans l'écriture des entrées sont rares [101]. Cela nous permet donc de supposer que notre hypothèse de départ est valide.

De plus cette invariance relative dans le temps des entrées et leur description dans le code source nous permet de dire que les entrées associées au même événement suivront toujours le même schéma. Et ainsi si deux entrées partagent la même partie fixe alors elles décrivent le même événement.

Cependant cette détection des parties fixes n'est pas immédiate car, comme expliqué précédemment, le champ message ne suit pas de norme. Les parties fixes et variables ne sont

donc pas distinguées par un signe particulier dans les entrées ("=" par exemple). De plus, même si le nombre de services systèmes et de composants est faible, un service système ou un composant peut écrire des entrées correspondant à plusieurs dizaines d'événements différents. Enfin, le volume total des entrées dont nous disposons (plus de 600 millions d'entrées pour la période analysée) empêche toute analyse manuelle exhaustive. Dans la suite, nous considérons uniquement la partie message des entrées pour associer un événement à l'entrée. Le mot "entrée" décrira donc uniquement le champ message de l'entrée.

Nous allons maintenant décrire les contraintes à respecter pour la construction de la solution.

5.4.2 Contraintes

Nous souhaitons construire un analyseur de journaux qui puisse analyser l'ensemble des entrées écrites en *temps réel* même lors de périodes de très forte activité.

Nous définissons la notion de *temps réel* comme le temps maximal nécessaire au traitement d'une entrée pour être capable de traiter l'ensemble des entrées produites par seconde en une seconde. Autrement dit, nous souhaitons que notre analyseur soit capable d'analyser les entrées de la précédente seconde en moins d'une seconde pour ne pas ajouter un délai sur le traitement des entrées de la prochaine seconde. En effet, si l'analyseur ajoute un court délai à chaque nouvelle seconde, ce court délai risque de se cumuler et d'entrainer au fur et à mesure un délai de plus en plus important dans le traitement des nouvelles entrées.

Il nous semble important d'être capable d'associer en *temps réel* les entrées et les événements car notre solution, LogFlow, peut être utilisée de manière interactive mais aussi de manière *préventive*. Si un utilisateur classe un ensemble d'événements comme intéressant car il présente les causes racines d'une défaillance par exemple, nous voulons être capable de dire en *temps réel* à cet utilisateur que cet ensemble d'événements est survenu sur le système pour lui permettre de mettre en place des actions. La détection de cet ensemble d'événements rend obligatoire l'analyse en *temps réel* de *l'ensemble* des entrées, même en cas de forte activité, pour être capable de détecter les événements correspondants aux événements présents dans l'ensemble choisi par l'utilisateur.

Pour connaitre la fréquence d'écriture des entrées, la figure 5.7 présente la distribution de la probabilité de générer un nombre d'entrées donné par seconde. Cette probabilité est calculée en utilisant les données de DKRZ. Nous calculons chaque seconde le nombre d'entrées générées sur le système. Nous comptons ensuite le nombre de secondes passées à générer à un nombre d'entrées donné. Enfin, la probabilité est calculée en divisant le nombre de secondes passées à générer un nombre d'entrées donné par le nombre total de secondes. Pour plus de facilité, le nombre des entrées générées est regroupé en puissance de 10.

La première observation est que notre système génère majoritairement entre 10^3 et 10^4 entrées par seconde. Cependant, il existe une faible probabilité (0.02) de générer 10^5 entrées par seconde. Quelques situations exceptionnelles ont généré un nombre très élevé d'entrées (supérieur à 10^5). Nous considérons ainsi que pour être capable d'analyser la plupart des entrées en temps réel (plus de 98% du temps), un analyseur de journaux doit être capable d'analyser jusqu'à 10^5 entrées par seconde. Nous considérons que les situations générant plus de 10^5 sont marginales et correspondent à des situations exceptionnelles (redémarrage de l'ensemble des nœuds, dysfonctionnement général du réseau, etc.) et peuvent donc être

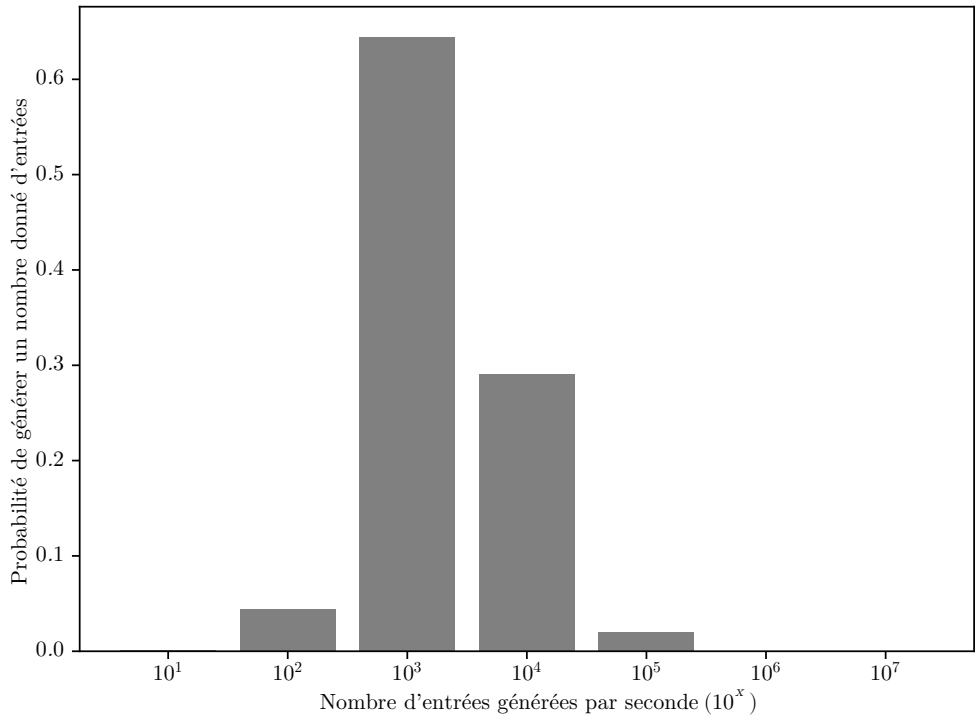


FIGURE 5.7 – Distribution de la probabilité de générer un nombre donné d'entrées par seconde.

analysés en prenant plus de temps et en ne respectant pas la contrainte du temps réel.

Nous décrivons maintenant plus précisément notre approche permettant de détecter les parties fixes et variables des entrées.

5.4.3 Découverte des parties fixes et variables dans les journaux systèmes

Notre méthode d'analyse de journaux repose sur la séparation des parties fixes et des parties variables des entrées. Une fois cette séparation faite, nous considérons que les entrées partageant les mêmes parties fixes appartiennent au même événement. Nous commençons par décrire l'algorithme proposé en utilisant un exemple puis nous explicitons ensuite les avantages de l'algorithme proposé.

Description de l'algorithme

L'algorithme que nous proposons est découpé en 4 étapes : i) découper les entrées en mots, ii) remplacer les mots par un descripteur, iii) construire des vecteurs de comparaison qui mesure le degré de similarités entre les entrées contenant le même nombre de mots, iv) déduire les schémas des entrées à partir des vecteurs de comparaison.

La première étape est de découper les entrées en mots. Pour cette étape, un séparateur évident est l'espace. Nous considérons aussi les caractères " :" et "=" comme séparateur. En effet, nous avons remarqué qu'ils peuvent être utilisés pour séparer les parties variables et fixes du message. Par exemple, pour une entrée contenant "user=Felix", "user" est une partie fixe évidente alors que "Felix" est une partie variable.

Une fois cette étape de découpe réalisée, nous devons être capables de comparer deux entrées entre elles. L'utilisation d'une égalité stricte est une première solution. Cependant,

elle est trop restrictive. En effet, considérons deux adresses mémoires "0x0c35685d" et "0x200da20c". Dans le cas d'une égalité stricte, ces deux adresses sont vues comme deux mots différents au même titre que "0x0c35685d" et "Felix" par exemple. Elles seront donc considérées comme une partie variable du schéma de l'entrée. Cependant, il semble préférable de considérer que ces deux adresses mémoires ne sont pas des parties variables *quelconques* mais des parties variables contenant des adresses mémoires.

Pour réaliser cette égalité *intelligente*, nous pouvons demander à l'utilisateur de rédiger des règles pour reconnaître l'ensemble des mots remarquables tels que les adresses mémoires, les adresses IP, les noms d'utilisateur, etc. Cependant, nous nous sommes fixés comme contrainte d'avoir un système entièrement automatisé sans intervention de l'utilisateur. Nous choisissons donc d'associer automatiquement chaque mot à un descripteur qui représente et décrit le mot.

Ce descripteur suit les règles suivantes : si le mot est uniquement composé de lettres, le descripteur est le mot lui-même, si le mot est un chiffre ou un nombre, le descripteur est "NB". Pour tous les autres cas, nous utilisons un vecteur de 5 entrées. Les 4 premières entrées sont des valeurs booléennes (0 ou 1) qui décrivent la présence d'un type de caractère : caractère numérique, caractères majuscule, caractère minuscule, et caractère non alphanumérique. La dernière entrée est la taille du mot. Dans le cas des adresses mémoires le descripteur associé sera $(1, 0, 1, 1, 10)$, et pour "Felix" le descripteur associé sera "Felix". Cela nous permet donc de pouvoir différencier automatiquement les adresses mémoires du mot "Felix".

Une fois que tous les mots de l'entrée sont remplacés par un descripteur, nous pouvons utiliser une égalité stricte : deux mots sont considérés comme égaux si leurs descripteurs sont les mêmes.

L'étape suivante est de construire un vecteur de comparaison pour déterminer les parties fixes et variables de chaque entrée. Pour une entrée donnée, son vecteur de comparaison compte le nombre de fois où le même descripteur apparaît à la même position dans les autres entrées de même taille. Une fois le vecteur de comparaison construit, nous pouvons extraire le schéma associé à une entrée utilisant la règle suivante : la partie fixe d'un schéma est composée du plus grand ensemble de descripteurs qui ont le même nombre d'occurrences selon le vecteur de comparaison.

Pour illustrer les différentes étapes, nous considérons les 6 entrées suivantes :

```
1) Temperature_Celsius changed from 55 to 54
2) Connection of user=R52 from Moon
3) Temperature_Celsius changed from 54 to 53
4) Temperature_Celsius changed from 52 to 54
5) Connection of user=B782 from Moon
6) Connection of user=Felix from Mars
```

Le découpage des entrées en mots donne le résultat suivant (chaque espace décrit un nouveau mot) :

```
1) Temperature_Celsius changed from 55 to 54
2) Connection of user R52 from Moon
3) Temperature_Celsius changed from 54 to 53
4) Temperature_Celsius changed from 52 to 54
5) Connection of user B782 from Moon
6) Connection of user Felix from Mars
```

L'utilisation des descripteurs nous fournit la représentation suivante :

```
1) (0,1,1,1,19) changed from NB to NB
2) Connection of user (1,0,1,0,3) from Moon
3) (0,1,1,1,19) changed from NB to NB
4) (0,1,1,1,19) changed from NB to NB
5) Connection of user (1,0,1,0,4) from Moon
6) Connection of user Felix from Mars
```

Nous pouvons ensuite calculer les vecteurs de comparaisons.

Pour expliciter ce calcul, la figure 5.9 détaille le calcul vecteur de comparaison pour la ligne 2). Pour garder une figure compacte, nous ne représentons que la comparaison avec la ligne 3), 5) et 6). La première étape (étape 0) initialise le vecteur de comparaison à $(0, 0, 0, 0, 0, 0)$ pour la ligne 2). Le vecteur est ensuite construit en comparant successivement la ligne 2) aux autres lignes. La première comparaison (étape 1) entre la ligne 2) et la ligne 3) montre qu'il n'y a aucun descripteur en commun entre les deux lignes. Le vecteur de comparaison n'évolue donc pas, sa valeur est de $(0, 0, 0, 0, 0, 0)$. La deuxième comparaison (étape 2) entre la ligne 2) et 5) indique qu'il a 5 descripteurs en commun entre les deux lignes (les descripteurs en commun sont en gras) : "Connection", "of", "user", "from", "Moon". Le vecteur de comparaison évolue donc et s'incrémente de 1 aux emplacements correspondants aux descripteurs en commun. Sa valeur est donc maintenant de $(1, 1, 1, 0, 1, 1)$. Enfin la dernière comparaison (étape 3) entre la ligne 2) et la ligne 6) indique qu'il y a 4 descripteurs en commun entre les deux lignes : "Connection", "of", "user", "from". Le vecteur de comparaison est incrémenté de 1 aux emplacements correspondant aux descripteurs en commun. Sa valeur est donc de $(2, 2, 2, 0, 2, 1)$. La construction du vecteur de comparaison est maintenant terminée sur cet exemple.

Le calcul des vecteurs de comparaison des autres entrées se réalise de la même manière. Pour l'entrée 1), son vecteur de comparaison est $(2, 2, 2, 2, 2, 2)$ car les entrées 3) et 4) incluent les mêmes descripteurs de mots. Le vecteur de comparaison est le même pour les entrées 3) et 4). Pour l'entrée 2), le vecteur de comparaison est donc $(2, 2, 2, 0, 2, 1)$. Le vecteur de comparaison de l'entrée 5) est identique. Le vecteur de comparaison de l'entrée 6) est $(2, 2, 2, 0, 2, 0)$.

Une fois l'ensemble des vecteurs de comparaison calculé, nous pouvons déduire les schémas associés à ceux-ci. De la même manière que précédemment, nous détaillons la déduction du schéma de l'entrée 2) à partir de son vecteur de comparaison à l'aide de la figure 5.9. La valeur est du vecteur de comparaison est de $(2, 2, 2, 0, 2, 1)$. La première étape consiste à calculer le nombre de fois où chaque descripteur apparait à la même position dans les autres lignes. Nous avons ici 4 descripteurs qui apparaissent à la même position dans 2 autres lignes, 1 descripteur qui apparait à la même position dans 1 autre ligne et 1 descripteur qui n'apparait à la même position dans aucune autre ligne. La déduction du schéma est ensuite réalisée en considérant que les parties fixes sont composées du plus grand nombre de descripteurs qui apparaissent dans le même nombre de ligne à la même position. Par conséquent dans cet exemple les 4 descripteurs qui apparaissent à la même position dans 2 autres lignes sont choisis comme parties fixes. Finalement, le schéma associé est donc le suivant : "Connection of user * from *" (les parties variables sont substituées par le joker "*").

La déduction des schémas à partir des vecteurs de comparaison des autres entrées se réalise de la même manière. Les entrées 2) et 5) partagent le même vecteur de comparaison. Le schéma déduit est donc le même "Connection of user * from *" . L'entrée 6) a un vecteur de comparaison légèrement différent. Cependant, le plus grand nombre de descripteurs qui apparaissent dans le même nombre de ligne à la même position reste le même que pour les entrées 2) et 5). Le schéma déduit est donc le même "Connection of user * from *".

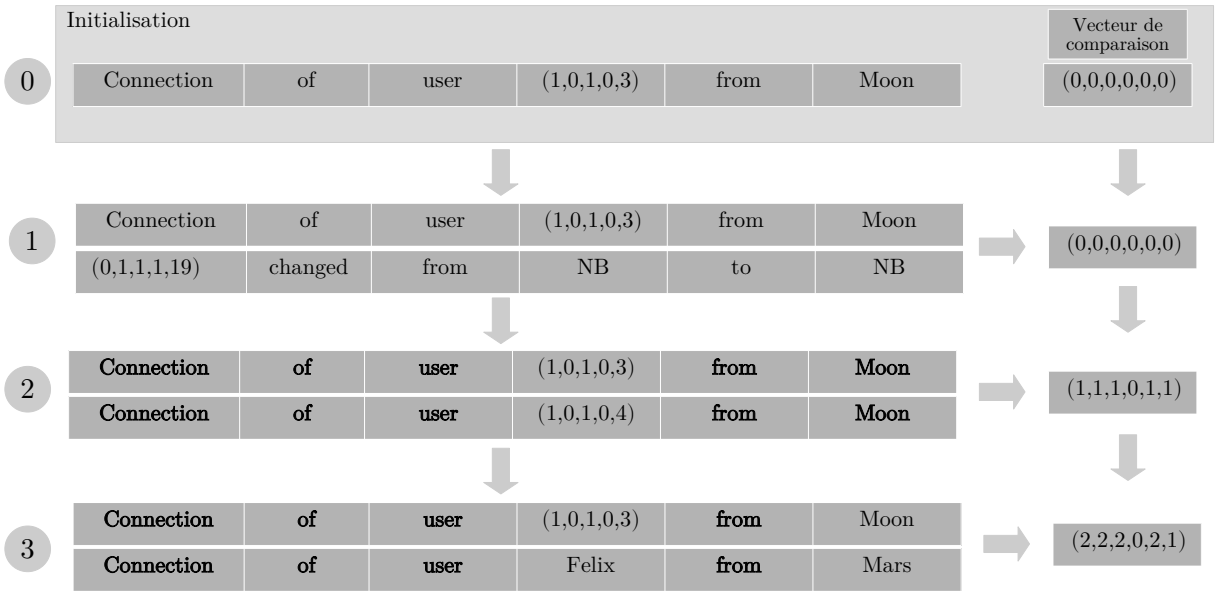


FIGURE 5.8 – Calcul d'un vecteur de comparaison pour une entrée.

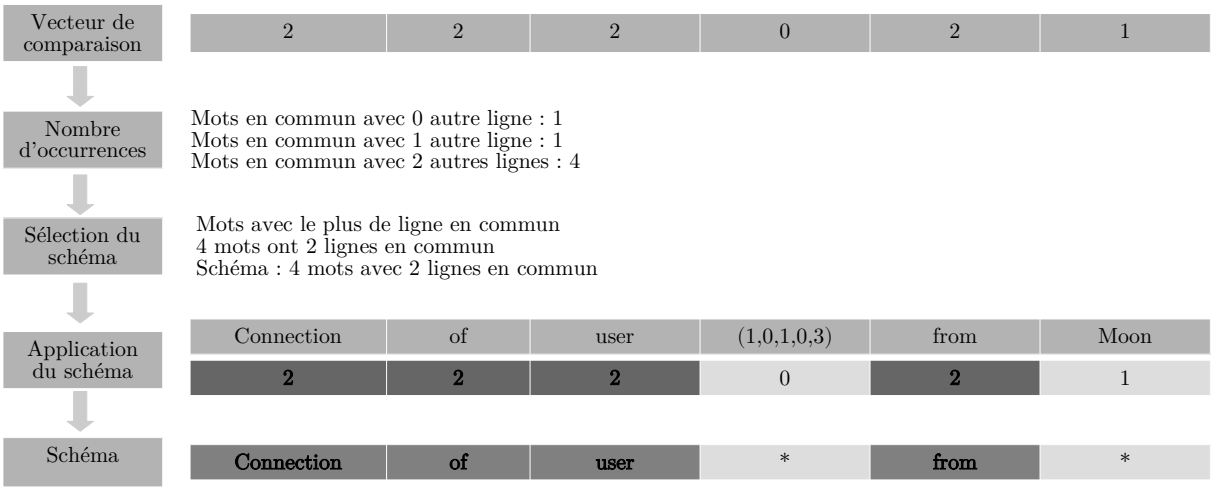


FIGURE 5.9 – Détermination du schéma associé à une entrée à partir du vecteur de comparaison.

Le vecteur de comparaison des entrées 1), 3) et 4) est identique. Le schéma déduit sera donc le même. Pour ces lignes, il y a 6 descripteurs qui apparaissent à la même position dans 2 autres lignes. Ils sont donc choisis comme partie fixe. Le schéma déduit est donc l'entrée elle-même "(0,1,1,1,19) changed from NB to NB"

Une fois l'ensemble des schémas déduits, les entrées associées avec le même schéma, et donc avec le même événement, sont attribuées au même numéro d'identifiant d'événement unique.

Il est à noter que si la valeur du vecteur de comparaison d'une entrée était de $(2, 2, 2, 5, 2, 1)$ par exemple, les parties fixes sélectionnées seront aussi les 4 descripteurs avec 2 lignes en commun et non l'unique descripteur avec 5 lignes en commun.

Notre algorithme, ainsi que les suppositions que nous avons réalisées sur la structure des journaux, nous permettent d'avoir une mise en œuvre efficace de notre solution comme nous le montrons dans le paragraphe suivant.

Mise en œuvre efficace

L'algorithme présenté nous permet de construire une implémentation simple et efficace de notre analyseur de journaux et il offre de multiples propriétés avantageuses.

Le premier avantage est la réduction de l'empreinte mémoire. En effet, en remplaçant les mots par un descripteur nous réduisons l'empreinte mémoire de l'algorithme (des longs mots comme les chemins d'accès à des fichiers sont remplacés par des vecteurs de 5 valeurs entières).

Le deuxième avantage est la possibilité d'utiliser un dictionnaire pour accélérer la construction des vecteurs de comparaisons. En effet, notre algorithme, tel que décrit précédemment, nécessite de comparer chaque entrées des journaux systèmes avec toutes les autres entrées. Pour éviter en pratique cette comparaison avec l'ensemble des autres entrées et gagner en temps d'exécution, nous pouvons utiliser un dictionnaire. Ce dictionnaire utilise une clé composée du descripteur et de sa position dans l'entrée. Chaque clé enregistre le nombre d'occurrences du descripteur à une position donnée. Le dictionnaire est créé lors de la lecture du fichier contenant les entrées. A chaque nouvelle entrée, les descripteurs de l'entrée sont créés et le dictionnaire est mis à jour en fonction de ces descripteurs et de leur position dans la ligne. Par conséquent, au lieu de devoir réaliser $n \times l$ comparaisons (n étant le nombre total d'entrées et l étant le nombre de mots dans une entrée) pour calculer le vecteur de comparaison d'une entrée (comme les exemples précédents le suggère), la construction du vecteur de comparaison pour une entrée ne demande que l accès aux dictionnaires.

Enfin, le dernier avantage est que nous supposons au départ que toutes les entrées qui correspondent au même événement ont la même taille. Cette supposition nous permet de construire simplement une version parallèle de notre solution. En effet, l'analyse des entrées de tailles différentes peut être réalisée dans des tâches totalement indépendantes qui peuvent être exécutées en parallèle. Par exemple, les entrées contenant 4 mots seront traitées dans un processus, celles contenant 5 mots dans un nouveau processus et ainsi de suite. Cette implémentation permet de tirer parti des capacités de parallélisation des processeurs actuels disposant de multiples cœurs de calcul et permet par conséquence de réduire le temps d'exécution nécessaire à la détection des parties fixes et variables.

Bien que l'algorithme présente de multiples avantages, il est nécessaire d'évaluer si les suppositions que nous avons réalisées sont réalistes. Pour cela, la prochaine partie présente l'évaluation des résultats de l'algorithme proposé sur plusieurs jeux de données de l'état de l'art ainsi que sur le jeu de données des journaux systèmes de DKRZ.

5.4.4 Résultats

L'évaluation des résultats de l'algorithme doit être faite selon deux caractéristiques : la précision et la rapidité d'exécution. En effet, dans un cas extrême, nous pourrions imaginer que l'ensemble des entrées soient traitées par des administrateurs. Cela nous donnerait une excellente précision (en considérant que les administrateurs ne se trompent jamais), mais un temps d'exécution très important empêchant un traitement dans en *temps réel*.

Nous allons nous intéresser dans un premier temps au temps de traitement nécessaire pour analyser l'ensemble des données, puis nous nous focaliserons ensuite sur la précision de l'algorithme proposé.

Temps de traitement

Le temps de traitement de la solution évaluée doit respecter la contrainte de *temps réel* que nous avons expliqué dans le paragraphe 5.4.2.

Pour évaluer cette contrainte, nous considérons que l'analyseur doit être capable d'analyser au minimum 10^5 entrées par seconde. Autrement dit, le temps maximal de traitement d'une entrée doit être de 0.01 ms par entrée. Ainsi, si nous considérons un temps maximal de 0.01 ms par entrée, le temps de traitement du jeu de données complet de DKRZ contenant 662 781 666 entrées doit être au maximum de $0.01 * 662781666 = 112$ *minutes*.

La mesure de temps est effectuée à partir du démarrage de l'analyseur jusqu'à la fin de l'écriture des résultats. L'ensemble des étapes des solutions est ainsi pris en compte en incluant les accès au disque nécessaire pour lire et écrire des données. Chaque mesure est répétée 3 fois et nous calculons ensuite la moyenne des 3 mesures. Il est à noter que le système utilisé pour la mesure est exclusivement dédié à celle-ci. Aucune autre application ne fonctionne sur celui-ci afin de ne pas perturber la mesure.

En complément de la mesure de temps de traitement sur le jeu de données provenant de DKRZ, nous incluons également la mesure sur les 2 autres jeux de données les plus volumineux présentés dans le tableau 5.3 provenant de l'étude proposée par Zhu et al. : `"Thunderbird"`, `"Windows"`. Nous analysons aussi les résultats en utilisant le jeu de données `"BlueGene/L"` car il est très souvent utilisé dans l'état de l'art [37, 38, 56, 57, 107]. Enfin, pour évaluer la pertinence de notre analyseur dans d'autres cas que les systèmes HPC, nous utilisons deux jeux de données provenant aussi de l'étude de Zhu et al. et issus de plateformes *Big data* : `"HDFS"` et `"Spark"`.

Nous analysons les temps de traitement mis par les analyseurs présentés précédemment dans le tableau 5.4 et par l'analyseur que nous proposons. Pour les analyseurs présentés dans le tableau 5.4, nous utilisons l'implémentation et les paramètres fournis par LogPAI [4].

Le tableau 5.5 fournit les temps d'exécutions des différents analyseurs de journaux. Le tableau ne présente qu'un seul analyseur en plus de LogFlow, Drain [45], car aucun autre analyseur de l'état de l'art présenté dans le tableau 5.4 n'a réussi à analyser notre jeu de données en moins de 48h, nous ne les faisons donc pas figurer dans le tableau. Le tableau présente deux versions de notre analyseur : LogFlow qui est la version multi-processus et LogFlow ST qui est la version mono-processus (les entrées de différentes tailles ne sont plus traitées parallèlement mais séquentiellement). En effet, aucun autre algorithme ne propose de version multi-processus, il nous semble donc équitable de comparer aussi les performances mono-processus de notre solution.

Le temps d'exécution sur le premier jeu de données `"BlueGene/L"` montre des temps d'exécution comparables entre notre solution (mono et multi-processus) et Drain. Le deuxième jeu de données `"Windows"` est analysé en 154 secondes par notre solution multi-processus et en 1166 secondes par notre solution mono-processus. Drain analyse ce jeu de données en 2100 secondes. Le troisième jeu de données `"Thunderbird"` est analysé en 145 secondes par notre solution multi-processus et en 799 secondes par notre solution mono-processus. Drain n'arrive pas à analyser l'ensemble du jeu de données en moins de 48 heures. Enfin, notre jeu de données `"DKRZ"`est analysé en 162 secondes par notre solution multi-processus et en 2388 secondes par notre solution mono-processus. Drain analyse ce jeu de données en 41h. Les résultats sont similaires concernant `"HDFS"` et `"Spark"`, notre solution multi-processus demande respectivement 30 secondes et 24 secondes, notre solution mono-processus requiert respectivement 249 secondes et 226 secondes alors que

4. https ://github.com/logpai/logparser commit 7be15f3

Jeu de données	Type	Nombre d'entrées	LogFlow	LogFlow ST	Drain
HDFS	*Big Data*	11M	30s	249s	2343s
Spark	*Big Data*	15M	24s	226s	3594s
Windows	Système	114M	154s	1166s	2100s
BlueGene/L	HPC	4.7M	9s	47s	50s
Thunderbird	HPC	211M	145s	799s	>48h
DKRZ	HPC	611M	162s	2388s	41h

Tableau 5.5 – Temps d'exécution des analyseurs de journaux.

Drain demande plus de 2000 secondes dans les deux cas.

La première conclusion est que l'algorithme représentant l'état de l'art actuellement en terme de précision, Drain, est incapable de respecter notre condition d'analyse en temps réel et ne peut donc pas être utilisé pour analyser des données de grande taille. Notre solution respecte cette condition d'analyse temps réel même en utilisant une version mono-processus. De plus, notre temps d'exécution reste stable en version multi-processus quel que soit la taille du jeu de données car nous sommes limités par les performances du disque dur et non par les performances du processeur ce qui explique un temps d'exécution comparable pour `"Thunderbird"`, `"Windows"` et `"DKRZ"` alors les jeux de données ne sont pas de la même taille. Lors de l'utilisation de la version mono-processus, les performances du processeur deviennent l'élément limitant. Il est à noter aussi que le temps d'exécution de notre solution mono-processus n'est pas directement proportionnel aux nombres d'entrées grâce aux optimisations réalisées par l'utilisation des dictionnaires ou des descripteurs par exemple. Ce qui explique que le temps d'analyse de `"Thunderbird"` soit inférieur au temps d'analyse de `"Windows"` alors que `"Thunderbird"` contient deux fois plus d'entrées.

Nous devons mentionner que Drain présente l'avantage d'être une méthode *en ligne*. Cela signifie que l'algorithme peut découvrir des nouveaux schémas sans avoir besoin d'analyser à nouveau l'entièreté des journaux. Notre analyseur est une méthode *hors ligne*. Elle doit analyser l'ensemble des journaux pour découvrir des nouveaux schémas. Bien que Drain présente un avantage sur ce point, nous pouvons nuancer cet avantage. En effet, les schémas dépendent des logiciels et services présents. Un service donné ne peut produire qu'un nombre fini de schémas différents et ces schémas n'évoluent que peu avec le temps [101]. Nous pouvons donc considérer que nous pourrons observer l'ensemble des schémas générés par chaque service si nous avons un historique de données suffisamment long. Lors d'une mise à jour des systèmes et/ou logiciels déjà présents, introduisant de nouveaux schémas, une nouvelle analyse sera nécessaire. Bien que Drain soit une méthode *en ligne*, son temps d'exécution le rend incompatible avec une analyse en *temps réel* des entrées des journaux systèmes pour un système tel que DKRZ.

Précision

La précision des détections est aussi un élément important à prendre en compte. En effet, si un analyseur n'est pas capable de détecter correctement les événements associés aux entrées alors son utilité est compromise. Bien que le paragraphe précédent conclue qu'il n'existe, à notre connaissance, pas d'analyseur dans l'état de l'art capable d'analyser notre jeu de données en temps réel, il est nécessaire de comparer la précision de notre solution avec celle obtenue par Drain. Pour rappel, Drain est la méthode obtenant la meilleure précision moyenne sur l'ensemble des jeux de données testés par Zhu et al. [108].

Analyseur	HDFS	Spark	Windows	BlueGene/L	Thunderbird
LogFlow	1	0.920	0.691	0.950	0.986
Drain	0.998	0.920	0.997	0.963	0.956

Tableau 5.6 – Précision des analyseurs de journaux.

La précision obtenue est calculée avec la méthode et les données annotées proposées par Zhu et al. [108]. La précision est donc une précision stricte comme expliqué dans le paragraphe 5.3.1, un événement détecté est associé à un et un seul événement annoté. Si nous détectons deux événements pour un événement annoté, l'événement ayant le moins d'entrées associées à l'événement annoté sera considéré comme faux. Nous exécutons notre analyseur de journaux et Drain sur chacun des jeux de données utilisés précédemment et nous utilisons les événements détectés par chacune des solutions pour calculer la précision.

Le tableau 5.6 détaille les résultats obtenus. Nous voyons que sur les jeux de données `"HDFS"`, `"Spark"`, `"BlueGene/L"` et `"Thunderbird"` nos résultats sont similaires à ceux obtenus avec Drain. Cependant sur le jeu de données `"Windows"`, nos résultats sont inférieurs. Ceci est dû au schéma associé aux entrées suivantes :

```
1) Session: 11152_142416105 initialized by client SPP
2) Session: 30546354_3363894584 initialized by client WindowsUpdateAgent
```

Notre solution n'attribue qu'un seul schéma pour les deux entrées et par conséquent les regroupent sous le même événement :

```
Session: * initialized by client *
```

Les annotations faites par les auteurs suggèrent que les deux entrées doivent être séparées en deux événements distincts :

```
Session: * initialized by client SPP
Session: * initialized by client WindowsUpdateAgent
```

Si nous considérons le fait de regrouper ces deux entrées sous le même événement comme correct, la précision de notre solution augmente de 0.691 à 0.996 et atteint des performances similaires à Drain.

5.4.5 Synthèse

Nous avons vu qu'actuellement et à notre connaissance, il n'existe pas d'analyseur de journaux dans l'état de l'art capable de traiter en temps réel le volume de données généré le système HPC étudié dans cette thèse. Notre proposition d'analyseur exploite la supposition suivante : si des entrées appartiennent au même événement alors elles doivent être de même taille. Cette première supposition nous permet de développer une solution multi-processus qui tire parti des capacités de parallélisation grandissante des processeurs. De plus, la construction des descripteurs de mots et l'utilisation ensuite d'une simple égalité de vecteur pour la comparaison réduit nos besoins en calcul. L'ensemble de ces optimisations nous permettent de créer une solution capable de traiter environ 4 millions d'entrées par seconde.

Cette rapidité d'exécution n'entrave cependant pas la précision de notre solution. En effet, sur 5 jeux de données, nous atteignons voire dépassons les résultats de l'état de l'art

dans 4 cas. L'analyse des résultats du dernier cas montre que les résultats sont fortement réduits à cause de la séparation d'un événement en deux-sous événements. En considérant cette séparation comme juste, notre précision atteint la valeur de l'état de l'art. Ces résultats calculés sur 3 types de jeux de données (HPC, Système d'exploitation et *Big data*) montrent aussi que la solution proposée peut être généralisée à d'autres types de journaux sans perdre en précision.

L'interprétation des résultats de la précision dépend fortement de l'utilisation envisagée de l'analyseur de journaux. En effet, dans le cas de l'utilisation des événements détectés par une solution automatique, celle-ci peut s'adapter à une détection imparfaite, avec par exemple une classe découpée en deux sous-classes. Nous montrerons dans le paragraphe 5.5.3 que malgré des différences dans la détection des événements entre notre solution et Drain, la méthode utilisant de l'apprentissage machine que nous proposons permet d'atteindre les mêmes résultats en termes de prédiction en utilisant les résultats fournis par Drain ou les résultats fournis par notre analyseur de journaux.

5.5 Détection des relations causales entre les événements

La détection des événements dans les journaux systèmes est la première étape pour permettre la mise en évidence des relations causales entre ces événements. Il est nécessaire ensuite de construire une solution pour la détection de ces relations.

Cette détection repose sur l'utilisation d'un LSTM et d'une couche d'attention. L'apprentissage est réalisé en apprenant à prédire l'événement actuel à partir des événements précédents dans les journaux systèmes. Si la prédiction réalisée est correcte, alors nous pouvons penser qu'il existe des liens entre les événements précédents et l'événement actuel permettant de réaliser cette prédiction.

Nous commençons par décrire LogFlow puis nous expliquons le flux de traitement des événements à travers celui-ci. Ensuite nous analysons les résultats obtenus et nous montrons finalement une représentation des relations causales détectées par LogFlow sur un cas réel.

5.5.1 Description de LogFlow

Notre solution repose sur la prédiction de l'événement actuel à partir des événements précédents pour détecter des relations causales entre ceux-ci. Si la prédiction réussie, nous pouvons donc en déduire qu'il existe des liens causaux entre les événements précédents et l'événement actuel.

Pour réaliser cette prédiction, notre solution utilise un algorithme d'apprentissage machine. Lors de la phase d'apprentissage, l'entrée de l'algorithme est composée des événements précédents et la sortie correspond à l'événement actuel. Par conséquent, l'algorithme apprend les corrélations entre l'événement actuel et les événements précédents. Lors de la phase d'utilisation, grâce aux corrélations précédemment apprises, l'algorithme pourra prédire l'événement actuel à partir des événements précédents. S'il réussit, nous pourrons alors déterminer quels sont les liens entre ces événements.

La figure 5.10 présente les 3 grandes étapes de LogFlow : la première étape est d'utiliser l'analyseur de journaux pour détecter les schémas dans les entrées et associer à chaque entrée un événement, la deuxième étape pré-traite les données obtenues pour faciliter

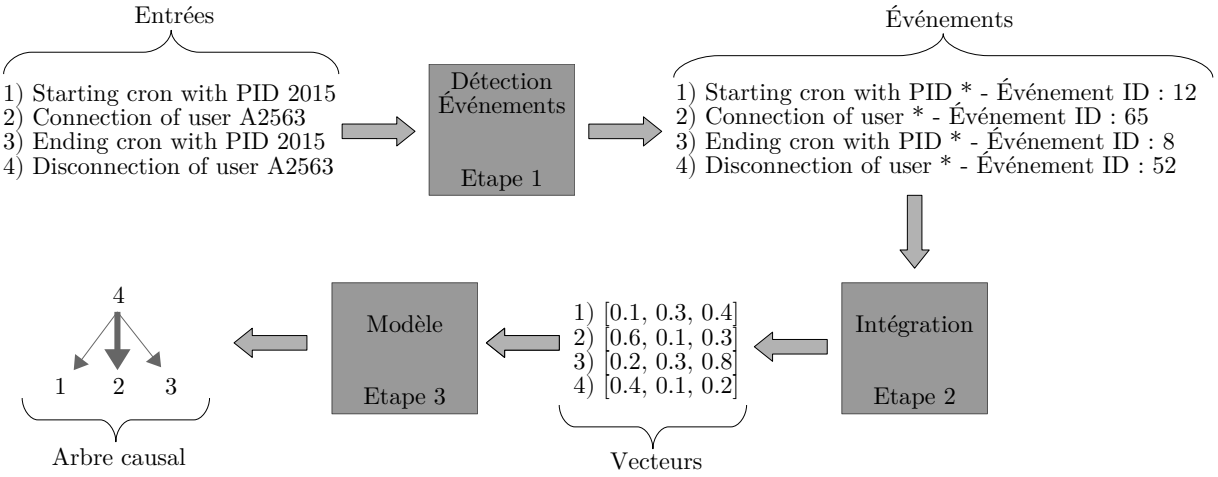


FIGURE 5.10 – Présentation des 3 étapes de fonctionnement de LogFlow.

le travail du LSTM qui représente la troisième étape. Le LSTM est ensuite utilisé pour déduire les relations causales entre les événements. Nous décrivons maintenant de manière détaillée ces 3 étapes. Nous détaillons aussi les choix réalisés pour construction du LSTM et de la couche d'attention.

Analyse des journaux systèmes

La première étape est d'utiliser l'analyseur de journaux pour déterminer les événements associés aux entrées. Cette étape est décrite dans la section 5.4.

Transformation des événements en vecteur

A la fin de l'étape de l'analyseur de journaux, nos données sont composées d'une suite d'événements. Chaque événement est décrit par un identifiant unique. En effet, comme nous l'avons expliqué dans la section 5.3.2, bien que cette suite soit adaptée pour une analyse manuelle des données, elle n'est pas adaptée à son traitement par un algorithme d'apprentissage machine. En effet, cette suite ne représente pas les liens possibles entre les événements. Il est ainsi impossible de dire que les événements 2 et 3 sont plus proches que les événements 3 et 10.

Pour représenter les liens possibles entre les événements, une étape d'intégration (*embedding*) est réalisée. Elle doit permettre de représenter que par exemple "serveur" et "ordinateur" sont deux mots proches alors que "serveur" et "chat" sont deux mots éloignés. Ainsi cette nouvelle représentation va permettre à l'algorithme d'apprentissage machine de construire des modèles plus représentatifs et plus fiables.

Pour réaliser cette étape, LogFlow repose sur l'utilisation de l'algorithme de word2vec [64]. Grâce à l'utilisation de cet algorithme, nous pouvons construire une représentation préservant le sens des événements à partir de la suite d'identifiants uniques. Cette représentation permet d'effectuer des opérations algébriques sur les vecteurs obtenus ce qui est important lors de la phase d'apprentissage du modèle d'apprentissage machine. Elle permet par exemple de calculer $roi - homme + femme = reine$.

Algorithme d'apprentissage machine

La taille de nos données, plus de 600 millions d'entrées, nous dirige vers l'utilisation d'un algorithme d'apprentissage profond (section 2.3.3). Pour rappel, l'analyse des journaux systèmes est similaire à l'analyse des séries temporelles, chaque nouvelle entrée dans les journaux correspond à un point de données ordonné chronologiquement. Nous choisissons donc d'utiliser un réseau de neurones spécialisés dans l'analyse des séries temporelles : le LSTM.

Cependant, la détermination des liens entre les événements n'est pas directement possible avec les LSTM car ce sont des algorithmes *boîtes noires*. Il est ainsi nécessaire d'ajouter une couche d'attention (ou module d'attention par la suite) pour permettre d'interpréter le LSTM et déterminer les liens entre les événements.

Pour obtenir les meilleurs résultats possibles, l'implémentation de la couche d'attention doit être réalisée en fonction du type de réseau. Le choix des LSTM nous dirige donc vers des couches d'attention spécialisées pour ce type de réseau. De nombreux travaux [59, 72, 90] proposent des approches différentes pour spécialiser les couches d'attention au LSTM. Nous choisissons d'utiliser la solution *Temporal Attention-Gated Model* (abréviée TAGM par la suite) proposée par Pei et al. [72]. Le choix repose à la fois sur la facilité de développement de la solution proposée et sur ces bons résultats dans différents jeux de données de référence [72].

La couche d'attention fournit les poids respectifs de chaque entrée dans la prédiction actuelle. Plus un poids est élevé, plus l'entrée correspondant à ce poids a de l'importance dans la prédiction actuelle.

Construction des arbres causaux

Une fois que l'algorithme d'apprentissage machine et la couche d'attention nous fournissent l'importance des différentes entrées dans la prédiction actuelle, nous pouvons remonter dans les journaux afin de découvrir l'ensemble des relations causales. La première étape est de sélectionner un événement `E`. Nous demandons ensuite à notre solution de prédire cet événement `E` à partir des événements précédents. Si la prédiction est juste, alors notre solution a trouvé des liens entre l'événement `E` et les événements précédents. Nous pouvons ensuite utiliser les poids fournis par la couche d'attention pour savoir quels sont les événements ayant le plus d'importance (et donc de lien) dans la prédiction de `E`. Une fois que ce ou ces événements sont mis en évidence, les premières relations causales sont déterminées. Nous recommençons ensuite à prédire les événements précédemment mis en évidence en déterminant quels sont les événements important dans leurs prédictions et ainsi de suite. A la fin, nous obtenons donc un arbre avec comme racine l'événement `E` et l'ensemble des branches de l'arbre représente les relations causales entre les événements mises en évidence successivement.

Les différents éléments de la solution étant décrits, nous pouvons maintenant décrire le fonctionnement de la solution dans son ensemble.

5.5.2 Fonctionnement de LogFlow

LogFlow repose donc sur une solution prédictive utilisant un algorithme d'apprentissage machine pour en déduire les relations causales entre les événements. Comme pour toute utilisation d'un algorithme d'apprentissage machine, il est nécessaire de réaliser une étape d'apprentissage et par conséquent de construire un jeu de données permettant cet

apprentissage. Une fois cette étape réalisée, il est nécessaire de savoir comment interpréter les résultats de la couche d'attention, et en particulier de savoir comment choisir les entrées avec un poids assez significatif pour être considérées comme ayant une relation causale avec l'entrée actuelle. Enfin, la dernière étape est de construire un arbre permettant une visualisation simple des relations causales.

Nous allons maintenant décrire les solutions mises en place pour ces trois problématiques : la construction du jeu de données, l'interprétation automatique des résultats de la couche d'attention et la construction de l'arbre. Nous terminerons en décrivant le flux de traitement entier de LogFlow.

Création du jeu de données et adaptation du modèle

Notre jeu de données provenant du système HPC de DKRZ contient 661 millions d'entrées qui correspondent aux journaux systèmes de l'ensemble des nœuds de calcul de janvier à octobre 2018. Le jeu de données est d'abord traité par l'analyseur de journaux pour regrouper les entrées décrivant un événement similaire. Cette première étape met en évidence 1488 événements différents.

Une fois cette étape réalisée, l'algorithme de Word2vec est utilisé pour transformer les identifiants uniques des événements en vecteur. Une fois la transformation réalisée, nous obtenons donc un jeu de données de 661 millions de vecteurs. Par la suite, nous utiliserons le mot *événement* pour décrire indistinctement l'entrée présente dans les journaux systèmes, l'identifiant unique ou le vecteur associé à l'événement.

La construction du jeu de données montre que les événements n'apparaissent pas tous avec la même fréquence dans les journaux. Le tableau 5.7 présente la répartition des événements en fonction de leur cardinalité. La cardinalité est ici définie comme le nombre d'apparitions de l'événement en puissance de 10. Un événement ayant une cardinalité `C` apparaît donc 10^C fois dans les journaux. La cardinalité des événements varie entre 0 et 8. Par conséquent le plus rare des événements apparait moins de 10 fois dans les journaux systèmes alors que le plus fréquent des événements apparait plus de 100 millions de fois dans les journaux. Cette différence de cardinalité est un problème lors de l'utilisation d'algorithme d'apprentissage machine et plus particulièrement d'apprentissage profond [95, 14]. En effet, lorsque l'algorithme apprendra un exemple de l'événement le moins fréquent, il apprendra dans le même temps plus de 100 millions d'exemples du plus fréquent. Il devient alors évident que l'algorithme se focalise rapidement sur l'apprentissage des événements les plus fréquents pour réduire rapidement son taux d'erreur en ignorant progressivement les moins fréquents.

Cependant, nous souhaitons être capable de prédire *l'ensemble* des événements et pas uniquement ceux avec la plus grande cardinalité. Pour résoudre ce problème, nous proposons une solution utilisant l'approche du *Single Class Learning* [95]. Cette approche propose de construire un modèle pour prédire uniquement la classe avec le plus faible nombre d'exemples. Cependant, elle n'est adaptée qu'à la classification binaire, c'est-à-dire la classification en deux classes distinctes. Nous avons dans notre cas plus de 1400 classes différentes (une classe correspond à un événement). Nous ne pouvons donc pas raisonnablement construire un modèle par classe car cela conduirait à utiliser plus de 1400 modèles. Notre approche est de construire un modèle par cardinalité. Cela réduit le nombre de modèle à construire à 9.

Chaque modèle se charge ainsi d'apprendre à prédire les événements avec une cardinalité donnée `C`. Cependant, durant la phase d'apprentissage, l'entrée d'un modèle peut contenir n'importe quelle cardinalité d'événements. La spécialisation d'un modèle n'a lieu que

Cardinalité ($\times 10^x$)	0	1	2	3	4	5	6	7	8
Nombre des différents événements	339	174	150	103	504	161	44	12	1
Nombre d'entrées correspondantes	1k	7k	58k	450k	15M	25M	90M	425M	105M

Tableau 5.7 – Nombre des différents événements par cardinalité.

sur sa sortie. Durant la phase d'apprentissage, l'entrée du modèle ne sera composée que d'événements précédant un événement de cardinalité donnée `C`. De manière plus formelle, l'entrée du modèle spécialisé pour des événements avec une cardinalité C_a donnée sera composée d'une séquence d'événements $[e_o, e_1, \ldots, e_n]$ menant à un événement e_{n+1} avec une cardinalité C_a. Les événements e_i fournis en entrées n'ont pas nécessairement une cardinalité C_a.

Il est à noter que cette approche est uniquement possible dans notre cas d'utilisation car nous connaissons à l'avance *l'ensemble* des événements que nous souhaitons prédire. En effet, nous utilisons la prédiction pour être capable de détecter les relations causales avec les événements précédents et l'événement actuel. La prédiction est donc réalisée sur un événement sélectionné qui s'est produit dans le système et dont nous connaissons la cardinalité. Dans le cas de prédiction de manière générale, la prédiction se réalise sur le prochain événement qui ne s'est pas encore produit dans le système. Il n'est donc pas connu, ce qui empêche de pouvoir sélectionner le modèle correspondant à la cardinalité de l'événement pour réaliser la prédiction.

Interprétation des résultats de la couche d'attention

Après la phase d'apprentissage du LSTM, nous l'utilisons pour détecter les relations causales. Lors d'une nouvelle prédiction réalisée sur l'événement actuel, la couche d'attention fournit les poids correspondants à chaque entrée du LSTM. Chaque entrée correspond à un événement précédant l'événement actuel. Nous pouvons donc associer à chaque événement précédant l'événement actuel un poids. Plus le poids est élevé, plus l'événement est important dans la prédiction actuelle. Le choix peut être fait de montrer à l'utilisateur l'ensemble des poids associés aux événements et de le laisser déterminer quels sont les événements importants en lui demandant de fixer un seuil manuellement pour chaque nouvelle entrée. Comme expliqué dans le paragraphe 5.2.1, nous pensons que pour que la solution soit viable, elle nécessite d'être entièrement automatisée. Nous devons, par conséquent, déterminer de manière automatisée un seuil.

Cependant, les poids fournis par la couche d'attention varient pour chaque nouvelle entrée. Il n'est donc pas envisageable de fixer *à priori* un seuil pour l'ensemble des entrées.

Pour déterminer de manière automatisée le seuil permettant de mettre en évidence les poids significatifs dans la prédiction, nous calculons automatiquement un seuil selon la valeur des poids comme suit :

$$T = moyenne(W) + std(W)$$

où W est l'ensemble des poids fournis par la couche d'attention pour la prédiction d'une entrée et *std* est l'écart-type. Nous sélectionnons donc uniquement les événements avec un poids supérieur à la somme de la moyenne et de l'écart-type de toutes les valeurs des poids. Ce seuil nous permet de ne sélectionner que les valeurs remarquables. Il est calculé pour chaque nouvelle prédiction et donc chaque entrée à son propre seuil associé.

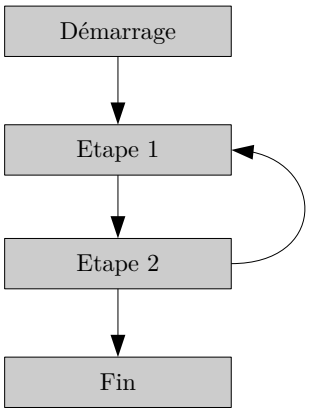


FIGURE 5.11 – Exemple de représentation agrégée d'un arbre causale.

Construction de l'arbre causal

Lorsque les entrées remarquables de la couche d'attention ont été sélectionnées, nous pouvons construire l'arbre symbolisant les relations causales. De la même manière que pour la sélection des valeurs de la couche d'attention, la construction de l'arbre doit être automatisée. Cependant, construire un arbre *exhaustif*, c'est-à-dire représentant l'entièreté des relations sélectionnées à partir de la couche d'attention, n'est pas souhaitable. En effet, en partant de l'hypothèse que seulement 2 lien causaux sont sélectionnés par entrée (nous aurions alors un arbre binaire), le nombre d'entrées analysées croît en fonction de $2^p - 1$ avec p la profondeur de l'arbre. Un arbre d'une profondeur 5, c'est-à-dire présentant les relations causales sur 5 niveaux depuis la racine, comprend déjà 31 nœuds. La profondeur de l'arbre risque donc d'être rapidement limitée par un seuil fixe pour garder un arbre d'une taille raisonnable et analysable facilement. Mais ce seuil peut nous empêcher de découvrir l'ensemble des relations causales.

Pour résoudre ce problème, nous remarquons dans un premier temps, qu'en pratique, l'arbre construit présente souvent des sous-arbres internes identiques à différents niveaux. Ceci est dû à la répétition des différentes actions ou étapes des services systèmes ou composants représentés par ces arbres. Pour représenter cette répétition, nous considérons le cas d'exécution d'une application. Cette exécution peut répéter plusieurs étapes identiques. Par exemple, considérons une application avec les différentes étapes suivantes : $Demarrage \rightarrow Etape\ 1 \rightarrow Etape\ 2 \rightarrow Etape\ 1 \rightarrow Etape\ 2 \rightarrow Fin$. Il est évident ici que l'arbre peut être agrégé en représentant l'étape 1 et 2 bouclant l'une sur l'autre. Cette représentation perd le nombre de répétition mais permet une représentation beaucoup plus compacte. La figure 5.11 montre cette représentation agrégée des étapes de l'application. Pour créer cette représentation agrégée, nous ajoutons donc une nouvelle relation causale dans l'arbre uniquement si celle-ci n'est pas présente. De la même manière, chaque nœud de l'arbre représente un événement unique. Toutes les entrées associées au même événement seront donc associées au même nœud de l'arbre. L'exploration de l'arbre s'arrête lorsqu'aucun nouveau nœud ou aucune relation causale ne sont ajoutés. Il est à noter que lorsque LogFlow prédit l'entrée sélectionnée pour obtenir les relations causales, si la prédiction est incorrecte alors nous arrêtons l'exploration pour cette entrée.

Flux de traitement

Le flux de traitement entier de LogFlow est le suivant. L'utilisateur sélectionne une entrée `X` dans les journaux systèmes. Cette entrée est associée à un événement `E` par l'analyseur de journaux systèmes. Cet événement `E` est ensuite associé à son vecteur grâce à l'algorithme de Word2Vec. Le modèle d'apprentissage machine prédit ensuite l'événement `E` à partir des `n` événements précédents. Si la prédiction est correcte, les relations causales seront données par les événements sélectionnés grâce au seuil sur les valeurs de la couche d'attention. L'exploration des relations s'exécutera ensuite sur les événements sélectionnés et ainsi de suite. Si la prédiction échoue, l'exploration s'arrête. Lorsque l'exploration ne trouve plus de nouveaux événements, elle s'arrête. L'arbre est alors construit et présenté à l'utilisateur.

Nous allons maintenant décrire les résultats obtenus par LogFlow en utilisant le jeu de données du système HPC de DKRZ.

5.5.3 Résultats

Pour que LogFlow soit utile, il doit être capable de prédire correctement l'ensemble des événements présents dans les entrées des journaux systèmes. Nous évaluons donc LogFlow sur ses résultats prédictifs. C'est à dire que nous voulons évaluer la capacité de LogFlow à prédire tous les événements présents dans notre jeu de données. Nous pouvons penser que si LogFlow parvient à prédire tous les événements alors nous serons capables de détecter les relations causales de n'importe quelle entrée présente dans nos journaux. Nous commençons par détailler les paramètres et données utilisés pour obtenir les résultats, ensuite nous présentons les métriques utilisées pour noter LogFlow, enfin nous finissons par exposer l'ensemble des résultats obtenus et une analyse de ces résultats.

Paramètres et données utilisés

Dans cette évaluation, nous considérons le jeu de données de DKRZ composé de 661 millions d'entrées associées à 1488 événements différents. Les 13 événements avec une cardinalité supérieure à 7 (apparaissant donc plus de 10^7 fois dans le jeu de données) sont ignorés. En effet, ces événements apparaissent si fréquemment qu'ils n'apportent pas d'information importante à l'utilisateur. Nous ignorons aussi les événements avec une cardinalité inférieure à 3 car les méthodes d'apprentissage profond ont besoin d'un grand nombre d'exemples pour fournir des résultats fiables [17, 83]. Dans la section 5.6, nous abordons des approches possibles pour intégrer ces événements dans notre solution. Nous construisons donc finalement 4 modèles (un par cardinalité 3, 4, 5 et 6) couvrant 721 événements et 130 885 952 d'entrées.

Nous configurons l'algorithme Word2Vec pour associer à chaque événement un vecteur de taille 20.

Nous utilisons 60% des données pour l'apprentissage et 40% pour le test. La génération des 2 jeux de données est réalisée en mélangeant le jeu de données initial et en le découpant en deux parties. De manière évidente, une fois que les événements mélangés sont sélectionnés, le jeu de données initial est utilisé pour fournir les événements précédents ceux sélectionnés comme entrée du modèle d'apprentissage profond.

Nous utilisons une fenêtre de 30 événements en entrée de notre modèle. Le modèle doit donc prédire le prochain événement à partir des 30 événements précédents. Nous évaluons l'influence de la taille de la fenêtre sur les résultats dans le paragraphe 5.5.3.

Classe	VP	FP	VN	FN	Rappel	Précision	Mesure F_1
`A`	58960	10	2524	25	0.99	0.99	0.99
`B`	365	900	10	20	0.95	0.29	0.44
`C`	256	25	40	765	0.25	0.91	0.39

Tableau 5.8 – Exemple de résultats illustrant l'utilisation des métriques.

Le LSTM est composé d'une couche unidirectionnelle avec une couche cachée de taille 50 suivie d'une couche d'attention et d'une couche pleinement connectée.

La condition que nous avons choisie pour arrêter le processus d'apprentissage est quand la *macro*-mesure F_1 n'augmente pas de plus de 0,01 par rapport à la valeur de l'itération précédente. En utilisant cette condition d'arrêt, l'entraînement du modèle dure approximativement 2 heures. Il est à noter qu'un entraînement pendant une période plus longue pourrait permettre d'améliorer légèrement les résultats.

Métriques

Notre modèle est un modèle multi-classes. Plusieurs métriques peuvent être utilisées dans ce contexte. Pour noter un modèle à 2 classes, les métriques classiques de précision, rappel et mesure F_1 sont couramment utilisées (section 4.3.1). Pour étendre ces métriques à un modèle multi-classes, nous calculons la précision, le rappel et la mesure F_1 pour toutes les classes. Nous pouvons ensuite construire les *micro*-métriques et les *macro*-métriques [105]. Les *macro*-métriques sont calculées en utilisant la moyenne sur la valeur de la métrique (par exemple le rappel) obtenue pour chaque classe. Les *micro*-métriques agrègent les résultats de prédiction de toutes les classes puis calculent les métriques.

Illustration des métriques

Pour illustrer la différence entre *micro* et *macro*-métriques, le tableau 5.8 montre un exemple utilisé pour le calcul de ces métriques. Nous considérons dans ce cas 3 classes : `A`, `B`, `C`. Pour chaque classe, le tableau donne la valeur des Vrais Positifs (VP), Faux Positifs (FP), Vrais Négatifs (VN) et Faux Négatifs (FN). La première remarque est que la classe `A` est majoritaire dans le jeu de données d'exemple. Le rappel de la classe `B` est élevé mais sa précision est faible. A l'inverse, la précision de la classe `C` est élevée mais son rappel est faible.

Le calcul des *macro*-métriques est le suivant :

$$
\begin{aligned}
macro - rappel &= \frac{0.99 + 0.95 + 0.25}{3} = 0.73 \\
macro - précision &= \frac{0.99 + 0.29 + 0.91}{3} = 0.73 \\
macro - mesure\ F_1 &= \frac{0.99 + 0.44 + 0.39}{3} = 0.61
\end{aligned}
$$

	Précision	Rappel
Macro-métriques	0.94	0.95
Micro-métriques	0.98	0.98

Tableau 5.9 – Qualité des prédictions de LogFlow sur le jeu de données de DKRZ.

Le calcul des *micro*-métriques est le suivant :

$$micro - rappel = \frac{58960 + 365 + 256}{(58960 + 25) + (365 + 50) + (256 + 765)} = 0.99$$
$$micro - précision = \frac{58960 + 365 + 256}{(58960 + 10) + (365 + 900) + (256 + 25)} = 0.98$$
$$micro - mesure\ F_1 = \frac{2 \times 0.99 \times 0.98}{0.99 + 0.98} = 0.99$$

Nous voyons que la valeur des *macro*-métriques dépend fortement des résultats de l'ensemble des classes alors que celle des *micro*-métriques dépend fortement des résultats de la classe majoritaire `A`. Ceci illustre la différence entre les deux types de métrique : les *macro*-métriques sont sensibles aux résultats de l'ensemble des classes alors que les *micro*-métriques sont sensibles aux résultats *globaux*. Ici les valeurs nous indiquent que globalement la solution est capable de prédire correctement un grand nombre d'exemples (*micro*-mesure F_1 de 0.99) mais que ces prédictions sont inégales selon les classes (*macro*-mesure F_1 de 0.61)

Evaluation des prédictions réalisées par LogFlow

Nous commençons cette section par donner les résultats généraux de LogFlow puis nous analysons ces résultats plus précisément en fonction du champ `"Catégorie"` et du champ `"Sévérité"` associés aux entrées. Ces champs sont décrits dans le paragraphe 5.3.1. Le tableau 5.9 présente les résultats de prédiction en fonction de la valeur des *macro*-métriques et des *micro*-métriques sur le jeu de données de DKRZ. Nous pouvons voir que LogFlow atteint de très bons résultats. La valeur des *micro*-métriques est de 0.98, la valeur de la *macro*-précision est de 0.94 et la valeur du *macro*-rappel est de 0.95. Ceci indique que LogFlow est capable de prédire *l'ensemble* des entrées de manière satisfaisante (valeur des *micro*-métriques) mais également que ces prédictions sont bonnes pour *l'ensemble* des événements (valeur des *macro*-métriques) et pas uniquement pour les événements avec un nombre d'exemples important.

Pour comprendre l'influence de notre choix de construire plusieurs modèles prédictifs en parallèle (un par ensemble d'événements ayant la même cardinalité) pour gérer le fait que certains événements apparaissent beaucoup moins souvent que d'autres dans les journaux, nous avons également construit une solution prédictive utilisant un seul modèle. Ce modèle obtient les mêmes résultats selon les *micro*-métriques, mais la valeur du *macro*-rappel est de 0.6 et la valeur de la *macro*-précision est de 0.59. Ces mauvais résultats sur les *macro*-métriques et les bons résultats sur les *micro*-métriques montrent que ce modèle est incapable de prédire correctement les événements qui apparaissent le moins fréquemment dans les entrées et démontrent le bénéfice d'utiliser plusieurs modèles.

Pour détailler les résultats obtenus par notre solution, le tableau 5.10 présente la distribution du nombre d'événements différents et du nombre d'entrées des journaux systèmes correspondant par rapport à la valeur de la *macro*-précision et du *macro*-rappel.

	Nombre d'événements		Nombre d'entrées	
Intervalle des *macro*-métrique	Précision	Rappel	Précision	Rappel
[0,0.1[	6 (0.8%)	4 (0.5%)	321	10
[0.1,0.2[	4 (0.5%)	0	11K (0.04%)	0
[0.2,0.3[	4 (0.5%)	3 (0.4%)	13K (0.04%)	10K (0.03%)
[0.3,0.4[	7 (0.9%)	4 (0.5%)	12K (0.04%)	38K (0.14%)
[0.4,0.5[	5 (0.7%)	6 (0.8%)	77K (0.29%)	95K (0.35%)
[0.5,0.6[	11 (1.5%)	13 (1.8%)	76K (0.28%)	39K (0.14%)
[0.6,0.7[	11 (1.5%)	13 (1.8%)	142K (0.53%)	351K (1.3%)
[0.7,0.8[	12 (1.7%)	17 (2.4%)	153K (0.57%)	53K (0.20%)
[0.8,0.9[	23 (3.2%)	23 (3.2%)	249K (0.9%)	405K (1.5%)
[0.9,1[	351 (49%)	390 (54%)	19M (71%)	20M (74%)
1	287 (40%)	248 (34%)	7M (26%)	6M (22%)

Tableau 5.10 – Distribution du nombre d'événements différents et du nombre d'entrées des journaux systèmes correspondant par rapport à la valeur de la *macro*-précision et du *macro*-rappel. Le nombre d'événements et d'entrées sont agrégés par intervalle de valeur des métriques.

Nous représentons la variation des mesures à l'aide d'intervalles. Le tableau montre que pour la plupart des événements la précision est supérieure à 90% et pour seulement 26 de ces événements la précision est inférieure à 50%. Un commentaire similaire peut être fait pour le rappel, sauf que dans ce cas seulement 17 des événements ont un rappel inférieur à 50%. Ces événements qui représentent 2,4% des événements au total correspondent à seulement 233k entrées dans les journaux systèmes, soit moins de 1%. Ces résultats montrent clairement que très peu événements peu fréquents ne sont pas correctement prédits par notre modèle.

Il est à noter que les 4 événements avec un *macro*-rappel inférieur à 0.1 ne correspondent qu'à 10 entrées alors que nous avons ignoré les événements avec une cardinalité inférieur à 3 (et donc un nombre d'entrées associées inférieur à 1000). Ceci s'explique par la sélection aléatoire qui est faite lors de la construction du jeu de données de test et du jeu de données d'apprentissage. Il peut arriver que 990 entrées d'un même événement soient sélectionnées pour l'apprentissage et seulement 10 pour le test (le ratio entre apprentissage et test est fixé globalement et non par événement).

Pour analyser plus en détails les résultats des prédictions, nous présentons une distribution de la *macro*-mesure F_1 en fonction du service qui a généré l'entrée et en fonction de la sévérité de l'entrée. Le tableau 5.11 résume les résultats présentés en fonction des intervalles de la *macro*-mesure F_1. Nous avons choisi de ne présenter que la *macro*-mesure F_1 dans ce tableau car les résultats précédents montrent qu'en général, le rappel et la précision ont des valeurs très proches. Les résultats montrent que pour la plupart des services et des sévérités, la *macro*-mesure F_1 est supérieure à 0.9. Trois services ont une *macro*-mesure F_1 inférieure à 0.9. Deux de ces services concernent des actions humaines (*slurmstepd*, *sssd*) et un est lié à des actions automatiques (*crond*). Ces caractéristiques rendent les événements associés difficiles à prédire car ils n'ont apparemment pas de liens avec des événements précédents. En ce qui concerne la sévérité des entrées du journal, seuls les événements avec une sévérité de type `alert` ont une *macro*-mesure F_1 inférieure à 0.9.

Pour mieux comprendre pourquoi seules les entrées des journaux systèmes avec une

macro-mesure F_1	Service	Sévérité
[50,60[	slurmstepd	/
[60,70[	crond	/
[70,80[	sssd	/
[80,90[	/	alert
[90,100]	useradd, hc, kernel, console-kit-daemon, fail2ban.server, ntpd, mcelog, logger, yum, fail2ban.action, rpcbind, groupmod, root, udevd, l-check, bootinv, fail2ban.filter, fail2ban.jail, fail2ban.database, inventory, postfix/master, postfix/postfix-script, kdump, acpid, postfix/smtp, post-fix/cleanup, dbus-daemon, login, CLM, anacron, check-ost, slurmd, fail2ban.actions, check_mountpoints.sh, cpus-peed, puppet-agent, sshd, smartd, init, crond	crit, err, notice, info, war-ning, debug, postfix/smtpd, post-fix/qmgr, iptables, ioadm

Tableau 5.11 – Moyenne de la *macro*-mesure F_1 pour chaque événement différent en fonction du service et de la sévérité de l'entrée associés à l'événement.

sévérité `alert` ont une *macro*-mesure F_1 faible, le tableau 5.12 montre la distribution des événements par classe de sévérité. Les mauvais résultats des entrées des journaux systèmes avec une sévérité `alert` s'expliquent par le faible nombre d'événements associés à cette classe (seulement 4) et les résultats relativement faible pour un événement.

Enfin, pour comprendre pourquoi de mauvais résultats sont obtenus avec quelques événements, nous montrons un exemple d'événement avec une *macro*-mesure F_1 inférieure à 0.1 :

```
pps_core: LinuxPPS API ver. 1 registered
```

Cet événement correspond au démarrage du logiciel LinuxPPS. Une étude manuelle des journaux systèmes montre que cet événement correspond toujours à la première entrée écrite lors du démarrage du logiciel. Il est donc difficile de prédire cet événement en utilisant uniquement les entrées précédentes car aucune ne permet de prédire que le logiciel sera démarré prochainement.

Les résultats présentés ici montrent que la solution que nous proposons, LogFlow, est capable de prédire avec précision les entrées des journaux systèmes dans la plupart des cas pour l'ensemble des événements. Les événements qui ne sont pas correctement prédits sont des événements qui, la plupart du temps, seraient difficiles à prédire (démarrage d'un logiciel par exemple). Les résultats montrent également que LogFlow traite efficacement le cas où le nombre d'entrées par événement est déséquilibré.

Macro-mesure F_1	alert	crit	debug	err	info	notice	warning
[0,0.1[	0	0	0	0	4	0	1
[0.1,0.0[	0	0	1	1	1	0	0
[0.2,0.3[	0	0	0	0	3	0	1
[0.3,0.4[	0	0	1	0	2	1	2
[0.4,0.5[	0	0	1	1	3	1	1
[0.5,0.6[	0	0	0	1	5	1	1
[0.6,0.7[	1	0	0	5	6	1	0
[0.7,0.8[	0	0	2	5	7	6	2
[0.8,0.9[	0	0	0	5	13	1	4
[0.9,1[	3	7	20	85	174	57	94
1	1	3	8	37	63	24	28

Tableau 5.12 – Nombre d'événements différents en fonction de la *macro*-mesure F_1 et en fonction de la sévérité des entrées associées aux événements.

	Précision	Rappel
Micro-métriques	0.95	0.95
Macro-métriques	0.98	0.98

Tableau 5.13 – Valeur des *macro*-métriques et *micro*-métriques values obtenues en utilisant l'analyseur de journaux Drain.

Comparaison avec Drain

La solution présentée utilise en entrée les événements détectés par un analyseur de journaux systèmes. Les résultats précédant sont obtenus en utilisant l'analyseur de journaux systèmes que nous avons proposé dans la section 5.4. Bien que l'analyseur proposé soit beaucoup plus rapide que les solutions existantes, il peut se révéler moins performant sur la détection d'événements comme le montre les résultats sur le jeu de données issu de Windows (paragraphe 5.4.4). En utilisant les résultats présentés par Xun et al. [108], Drain est actuellement l'analyseur de journaux systèmes présentant les meilleurs résultats sur l'ensemble des jeux de données évalués. Nous évaluons ici les résultats obtenus par la solution complète en utilisant Drain comme analyseur de journaux. Notre but est de connaître l'influence des résultats de l'analyseur de journaux sur les résultats de la solution entière.

De manière similaire au tableau 5.9, nous présentons les valeurs des *macro*-métriques et *micro*-métriques obtenues en utilisant l'analyseur de journaux Drain pour détecter les événements dans le jeu de données. Les résultats sont globalement les mêmes en utilisant les deux analyseurs. Nous pouvons seulement noter une légère différence de la *micro*-précision mais elle n'est pas significative car elle est du même ordre de grandeur que celui de la condition d'arrêt utilisée durant la phase d'entraînement (paragraphe 5.5.3).

Ce résultat tend à montrer que notre analyseur de journaux systèmes est suffisamment précis pour obtenir des bonnes prédictions. De plus, même si l'analyseur de journaux systèmes ne parvient pas à identifier correctement certains événements, le modèle d'apprentissage machine parvient à apprendre un modèle correct à partir de données d'entrée partiellement incorrectes.

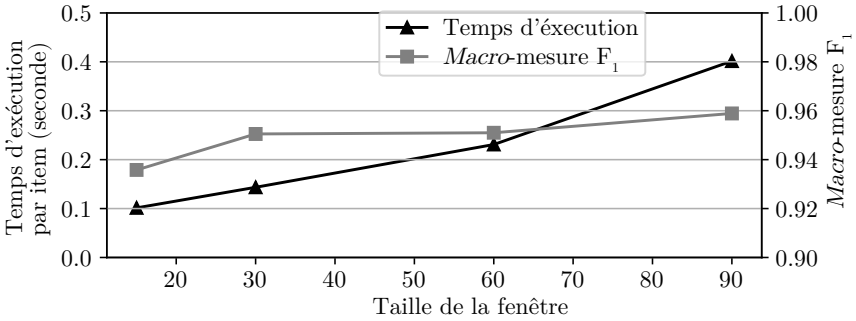


FIGURE 5.12 – Influence de la taille de la fenêtre sur la *macro*-mesure F_1 et sur le temps d'exécution.

Influence de la taille de la fenêtre

Un des paramètres pouvant influer sur les résultats de LogFlow est la taille de la fenêtre en entrée de la solution. En effet, plus cette fenêtre est grande plus LogFlow remontera loin pour chercher des relations causales et sera capable de prédire la prochaine entrée. Cependant, cette taille influe aussi sur le temps d'apprentissage et le temps d'exécution de LogFlow.

Pour évaluer l'influence de la taille de la fenêtre sur le temps d'apprentissage et sur les résultats finaux, la figure 5.12 montre la variation du temps d'apprentissage par item et la variation de la *macro*-mesure F_1 valeur en fonction de la taille de la fenêtre. Un item est ici défini comme un groupe de 128 entrées à prédire. La *macro*-mesure F_1 augmente significativement quand la taille de la fenêtre augmente de 15 à 30 mais augmente de moins de 0.01 ensuite.

D'un autre côté, le temps d'apprentissage croît significativement lors la taille de la fenêtre augmente. En utilisant ces conclusions, le meilleur compromis concernant la taille de la fenêtre est atteint lorsque la taille de celle-ci est fixée à 30.

D'après les résultats que nous avons présentés, la solution que nous proposons est donc capable de prédire la très grande majorité des événements présents dans notre jeu de données. Cependant, bien que ces résultats valident l'approche prédictive utilisée, ils ne reflètent pas l'utilité des relations causales mises en évidence. Nous allons donc dans la prochaine partie montrer un résultat concret obtenu sur un cas réel pour valider le fonctionnement de l'ensemble de la solution.

5.5.4 Présentation des arbres causaux obtenus par LogFlow

La finalité de la solution est de proposer un arbre des relations causales. A partir des prédictions réalisées précédemment, la couche d'attention fournit les poids indiquant l'importance des événements en entrée dans les prédictions. Dans cette partie, nous présentons un exemple d'arbre de relations causales obtenus à partir des données de DKRZ et nous étudions le temps nécessaire pour obtenir un arbre en pratique.

Exemple d'arbre

La figure 5.13 montre l'exemple d'un arbre obtenu en analysant les données de DKRZ. Cet arbre est obtenu en sélectionnant une erreur du logiciel Lustre comme événement initial à analyser. Nous avons manuellement supprimé les adresses IP et le nom des nœuds de calcul de l'arbre afin de garantir l'anonymisation des données requise pour la publication

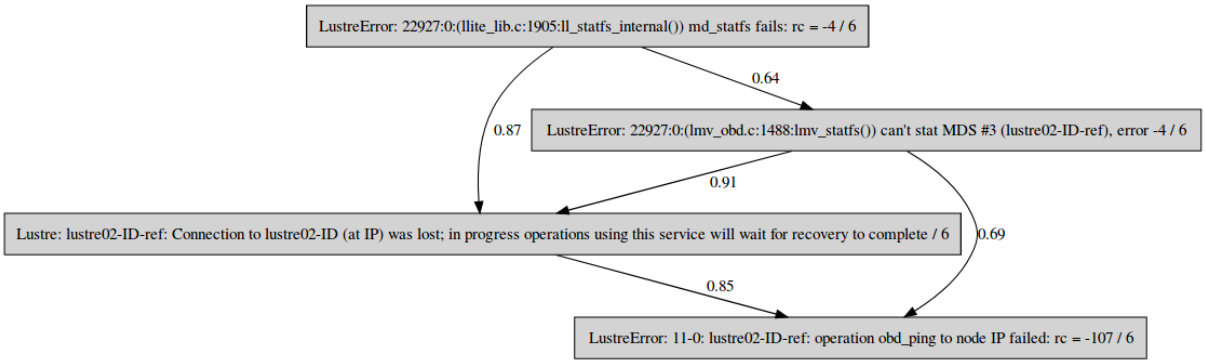


FIGURE 5.13 – Exemple d'un arbre des relations causales obtenu sur les données de DKRZ.

de cette thèse. Cette étape est uniquement réalisée pour la publication de l'arbre, elle n'est pas réalisée lors de l'apprentissage de la solution.

Les nœuds de l'arbre correspondent aux différents événements détectés à partir des entrées dans les journaux systèmes et les liens correspondent aux relations causales entre les événements. Les valeurs sur les liens correspondent aux poids fournis par la couche d'attention. Plus la valeur du poids est haute, plus le lien entre deux événements est fort. Pour construire cette arbre, notre solution commence par prédire l'entrée qui correspond à l'événement `"md_statfs fails: rc =-4"` en utilisant les événements précédents. La prédiction est correcte. Ensuite, le module d'attention fournit les poids associés aux événements précédents. Notre heuristique sur le seuil sélectionne deux entrées : `"can't stat MDS #3"` et `"Connect to lustre02-ID was lost"`. Nous répétons ensuite l'étape de prédiction avec les deux entrées sélectionnées.

L'arbre présenté ici peut nous aider à trouver la cause racine de la défaillance `"md_statfs fails: rc = -4"`. En effet, il montre que l'erreur doit être reliée à une erreur sur le réseau haute performance du nœud de calcul car le dernier nœud de l'arbre correspond à un ping qui échoue sur le nœud de calcul.

Il est à noter que les relations causales mises en évidence par cet arbre ont été validées par des experts locaux.

Temps d'exécution de la solution

Nous évaluons ici le temps requis par LogFlow pour produire l'arbre des relations causales. Une fois la phase d'apprentissage du modèle réalisée, l'utilisation pratique du modèle par l'utilisateur va généralement se dérouler sur l'ordinateur de celui-ci. Par conséquent, pour tenir compte des conditions matérielles d'utilisation de la solution, nous évaluons le temps pris par celle-ci pour construire un arbre des relations causales sur un ordinateur portable équipé d'un processeur Intel i7-7820HQ. Pour cette évaluation, le GPU est désactivé et la solution n'utilise que le processeur de l'ordinateur portable. Nous calculons la moyenne du temps total mis par la solution complète pour la création d'un arbre en incluant le temps nécessaire aux accès au disque, à l'analyseur de journaux, à la prédiction par le LSTM et à la création elle-même de l'arbre. La moyenne du temps total est calculée sur 100 arbres différents en divisant le temps total de création de l'arbre par la solution par le nombre d'événements présents dans l'arbre.

La création d'un arbre requiert moins de 10ms de temps d'exécution par événements présents dans l'arbre. La création de l'arbre précédant demande donc environ 50 ms. Nous pouvons donc conclure que l'analyse des relations causales à l'aide de notre solution est

assez rapide pour être utilisée dans un contexte interactif.

5.5.5 Synthèse

Nous avons montré à travers cette section que LogFlow permet la détection des relations causales entre les événements présents dans les journaux systèmes.

La solution proposée s'appuie sur une méthode prédictive utilisant un LSTM et une couche d'attention pour la détection des relations causales. L'analyse des résultats montre que nous sommes capables de détecter les relations causales d'une grande partie des événements se déroulant sur un système HPC.

De plus, la mesure du temps total nécessaire à la solution pour détecter les relations causales la rend compatible avec une utilisation sur un ordinateur portable dans un contexte interactif.

Enfin, l'analyse d'un exemple d'un arbre présentant les relations causales obtenues sur le jeu de données de DKRZ montre que les relations détectées et mises en évidences par la solution sont viables. En effet, l'arbre que nous présentons en exemple a été validé par des experts locaux comme ayant du sens.

Nous allons maintenant discuter des limitations induites par les suppositions faites pour réaliser la solution.

5.6 Discussion

Les suppositions réalisées pour rendre la solution totalement automatisée (*i.e.*, sans paramétrage humain) amènent des limitations. Nous allons dans cette partie discuter de ces limitations et de leur réelle influence dans les résultats.

La première limitation est au niveau de l'analyseur des journaux systèmes. Nous supposons que toutes les entrées appartenant au même événement font nécessairement la même taille et contiennent donc le même nombre de mots. L'utilisation de cette supposition nous permet de traiter facilement en parallèle l'ensemble des entrées et de réduire significativement notre temps d'exécution. Cependant cette supposition ne nous permet pas de rassembler des entrées avec un nombre variable de mots sous un même événement. Le jeu de données Proxifier qui est présenté dans le tableau 5.3 montre un exemple de deux entrées de tailles différentes qui peuvent être liées au même événement :

```
1) IP close, 745 bytes sent, 229 bytes received, lifetime 00:14
2) IP close, 1046 bytes (1.02 KB) sent, 5463 bytes (5.33 KB) received, lifetime 04:00
```

Ces deux entrées semblent indiquer un même événement : une connexion réseau a été fermée et l'entrée résume la durée et les données envoyées et reçues. Cependant, les deux entrées ne font pas la même taille. La deuxième entrée est plus grande car elle résume entre parenthèses la taille des données reçues et envoyées si la taille est supérieure à 1000 octets. Notre analyseur ne pourra pas rassembler dans ce cas les deux entrées sous un même événement, il détectera 2 événements distincts. Bien que cette détection ne soit pas *parfaite*, nous avons vu que l'utilisation par la suite d'une solution d'apprentissage profond permet de généraliser les connaissances apprises et d'obtenir une excellente précision globale malgré une détection imparfaite.

La deuxième limitation est l'élimination des événements *rares* du jeu de données d'apprentissage (les événements avec une cardinalité inférieure à 3) car la solution d'ap-

prentissage profond nécessite un minimum d'exemples pour fournir des résultats fiables. Ceci implique que la solution proposée n'est pas capable de détecter les relations causales pour un événement rare présent dans notre jeu de données. Or, il peut être intéressant pour un administrateur système de connaître les relations entre un événement rare et les événements qui le précédent. Pour mieux comprendre les conséquences de cette limitation, nous avons analysé manuellement les événements rares présents dans notre jeu de données. Nous avons identifié 3 catégories.

La première catégorie d'événements rares est due à un mauvais regroupement des entrées durant la phase d'analyse des journaux. Ce mauvais regroupement implique que certains événements devraient être fusionnés avec d'autres car ils sont similaires. Il est alors nécessaire soit de fusionner ces événements manuellement, soit d'apporter des modifications à l'analyseur de journaux pour améliorer les regroupements réalisés.

La deuxième catégorie d'événements rares correspond à des événements réellement rares, c'est-à-dire avec une faible probabilité d'apparition mais qui sont associés à des actions normales sur le système. Par exemple dans le cas de DKRZ, l'ensemble des événements associés au redémarrage d'un nœud sont des événements rares car le redémarrage d'un nœud dans un système HPC est très peu fréquent. Ce type d'événements rares peut être rapidement identifié par un utilisateur sans avoir besoin d'utiliser notre solution.

La troisième catégorie d'événements rares est liée aussi à des événements rares mais qui sont associés à des actions anormales ou à des dysfonctionnements sur le système. Contrairement à la deuxième catégorie, il peut être intéressant ici pour un utilisateur de connaitre les relations causales de ce genre d'événements avec les événements précédents. La solution proposée actuellement utilisant des réseaux d'apprentissage profond ne peut pas fournir des résultats fiables sur des événements avec peu d'exemples. Cependant, elle peut les détecter de manière *indirecte*. En effet, nous pouvons détecter l'ensemble des relations liant les événements adjacents à l'événement rare. Une fois l'ensemble des relations détectées, nous pouvons considérer que les relations liant les événements adjacents ne sont pas liées à l'événement rare. Nous pouvons donc supprimer l'ensemble des événements ayant une relation avec un événement adjacent. Cette suppression va permettre de mettre en évidence les événements précédents l'événement rare qui ne sont *à priori* reliés à aucun autre événement et qui peuvent aider à trouver la cause racine de l'événement rare.

Enfin, la dernière limitation de LogFlow est que les relations entre les événements ne sont mises en évidence que sur un seul nœud. En effet, la solution n'est actuellement pas capable de trouver les relations causales entre les entrées des journaux systèmes de différents nœuds du système. La figure 5.13 montre que le dernier événement de l'arbre est probablement lié à une défaillance sur le réseau d'interconnexion. Cependant, nous ne sommes pour le moment pas capables de corréler les entrées des journaux réseaux avec les entrées des journaux systèmes.

Une solution simple serait de fusionner l'ensemble des journaux dans un seul grand journal. La fusion de l'ensemble des journaux dans un seul grand journal risque de générer un bruit important dans les journaux. En effet, la fusion de l'ensemble des journaux systèmes des 3000 nœuds de calcul avec les journaux du réseau, du système de fichiers, etc, introduira un nombre important d'événements non corrélés entre des événements corrélés.

Pour résoudre ce problème, nous pourrions augmenter la taille de la fenêtre utilisée en entrée de la solution. C'est-à-dire augmenter le nombre d'entrées prises en compte par la solution pour prédire la prochaine entrée (30 actuellement). Cependant, le temps d'apprentissage du modèle augmente en fonction de la taille de la fenêtre. En augmentant de

manière importante la taille de la fenêtre, nous augmenterons donc de manière importante le temps d'apprentissage et le temps nécessaire à la découverte des relations causales lors de l'utilisation de la solution. Cette augmentation risque de réduire la possibilité d'utiliser la solution de manière interactive s'il est nécessaire d'attendre plusieurs minutes avant de pouvoir construire un arbre présentant les relations causales à partir des entrées. De plus, l'ajout de bruit dans les données par l'entrelacement des données provenant des autres nœuds du système risque aussi de réduire la capacité prédictive et les résultats de la solution.

Par conséquence, nous pensons qu'il est nécessaire d'exploiter la connaissance de la topologie du système HPC et des connexions entre les différents services inter-nœuds *dans* la solution pour garantir des résultats fiables. L'exploitation de cette connaissance pourrait permettre de naviguer intelligemment entre les journaux des différents composants du système au lieu de les agréger. Cette nouvelle solution représente une étape importante et requiert de nouveaux travaux.

5.7 Synthèse

Dans ce chapitre, nous proposons un outil qui fournit une vue augmentée mettant en évidence les relations entre les informations contenues dans les journaux systèmes. Cet outil génère un arbre représentant les relations causales entre les entrées présentes dans les journaux. Cet arbre permet de faciliter la recherche par l'opérateur des causes racines des différents événements survenant dans un système HPC.

Pour construire l'outil, deux étapes principales sont nécessaires : le regroupement des entrées des journaux systèmes décrivant un événement *similaire* et l'apprentissage des corrélations entre les événements précédemment mis en évidence.

La première étape est donc le regroupement des entrées des journaux systèmes décrivant un événement *similaire*. L'analyseur proposé dans ce chapitre permettant de regrouper les entrées repose sur la détection des parties fixe et variables des messages contenus dans les entrées. Les entrées partageant les mêmes parties fixes sont regroupées sous le même événement. En utilisant la supposition que les entrées décrivant le même événement sont de même taille (elles ont le même nombre de mots), nous pouvons construire un analyseur multi-processus. Grâce à cette supposition et à différentes optimisations réalisées telles que l'utilisation d'un dictionnaire et l'utilisation d'un descripteur pour décrire les mots présents dans les entrées, notre analyseur dépasse de plusieurs ordres de grandeur la vitesse d'exécution des analyseurs de l'état de l'art. De plus, la comparaison des résultats obtenus par notre analyseur et ceux obtenus par l'analyseur représentant l'état de l'art montre que nous arrivons à une précision similaire sur des jeux de données variés.

Une fois ce regroupement effectué, la détection des corrélations entre les événements est réalisée par un modèle d'apprentissage profond utilisant un LSTM et une couche d'attention. Les résultats montrent que ce modèle d'apprentissage profond nous permet de pouvoir découvrir les relations causales de l'ensemble des événements présents les journaux systèmes avec plus de 94% de précision. De plus, l'analyse d'un exemple d'arbre présentant les relations causales obtenues sur les données de DKRZ montre que les relations détectées ont du sens pour un administrateur et permettent de remonter efficacement à l'origine du dysfonctionnement. La validation de cet exemple d'arbre par des experts locaux permet confirmer l'utilité et la fiabilité de la solution.

Enfin, la solution construire est entièrement automatisée, elle ne nécessite pas de paramétrage humain. En complément, le temps d'exécution total de la solution pour la

détection des relations causales d'une entrée sélectionnée est inférieur à 10 millisecondes. Ceci nous permet d'envisager son utilisation par des administrateurs systèmes dans un contexte interactif.

En conclusion, la solution proposée dans ce chapitre permet de mettre en évidence de manière fiable les corrélations entre les événements dans les journaux systèmes. Son temps d'exécution et ses résultats la rendent compatible pour une utilisation interactive par des utilisateurs ou des administrateurs de système HPC.

Chapitre 6

Conclusion

Les besoins en calcul des applications scientifiques ne cessent de croître. Pour répondre à ces besoins, les systèmes HPC deviennent de plus en plus grands et ils intègrent de plus en plus de ressources qui peuvent être hétérogènes. La complexité accompagnant ces systèmes suit la même croissance.

La combinaison de l'ensemble de ces facteurs rend ces systèmes particulièrement sensibles aux défaillances. En effet, à cause de l'interdépendance de l'ensemble des ressources composant un système HPC, la défaillance d'une seule de ces ressources peut engendrer une défaillance de l'ensemble du système. De plus, la probabilité de subir une défaillance suit la même tendance que la taille des systèmes HPC : plus un système est grand et complexe, plus cette probabilité augmente [25].

Le besoin de haute performance de ces systèmes induit des conditions strictes à la fois sur l'architecture même du système mais également sur l'utilisation de ces systèmes. Ces conditions rendent ces systèmes rigides et peu capable de s'adapter naturellement aux différentes défaillances.

Au vu du coût humain et financier de ces systèmes (plusieurs millions d'euros), il apparaît naturel de s'intéresser à la question de la gestion des défaillances. En effet, ces défaillances peuvent influencer fortement la disponibilité du système et réduire son efficacité. Par conséquent, il est nécessaire d'entreprendre des actions pour mitiger leurs conséquences sur le système ou pour pouvoir, dans le meilleur des cas, les éviter en réalisant des actions préventives.

Il existe actuellement de nombreux travaux qui proposent des solutions variées pour assurer cette gestion des défaillances. Ces solutions reposent sur l'hypothèse que les défaillances surviennent dans le système et qu'il est nécessaire de trouver des actions pour limiter leurs conséquences. Une de ces solutions utilisée actuellement dans les systèmes est la sauvegarde régulière de points de reprise de l'application. Si une défaillance survient dans le système et que l'application s'arrête, elle rechargera son dernier point de reprise. Le temps d'exécution perdu sera donc uniquement celui jusqu'au dernier point de reprise et non celui depuis le début de l'application. Bien que cette solution soit simple à mettre en œuvre et permette de limiter les conséquences des défaillances, elle a des désavantages : transfert de données sur le réseau pour la création des points de reprise, espace de stockage nécessaire aux points de reprise, etc.

D'autres solutions sont actuellement développées et proposent d'analyser les défaillances pour mieux les comprendre et pouvoir soit les éviter à l'aide d'actions préventives soit améliorer les solutions actuelles de gestion des défaillances. Ces solutions s'appuient sur l'analyse des données de supervision fournit par les systèmes HPC. Ces données décrivent

l'état de santé du système. Elles contiennent donc de nombreuses informations intéressantes pouvoir permettre de comprendre les défaillances.

Cependant l'analyse de ces données de supervision n'est pas facile car elles sont le reflet d'un système HPC. Elles sont souvent complexes, de plusieurs types et de grande taille. L'ensemble de ces paramètres rend l'analyse manuelle de ces données fastidieuse et ardue. Les méthodes d'apprentissage machine peuvent aider à l'automatisation de cette analyse. En effet, il est envisageable de les utiliser pour permettre la découverte automatisée des corrélations entre les défaillances et les données de supervision. L'utilisation de ces corrélations peut ensuite permettre de mieux comprendre les défaillances et de pouvoir les prédire.

La mise en place de ces solutions d'apprentissage machine n'est pas une tâche facile. En effet, le fonctionnement de ces méthodes est complexe. Ainsi, pour obtenir des résultats fiables, il est nécessaire de choisir la meilleure méthode d'apprentissage machine en tenant compte des objectifs, des contraintes mais également des données.

Les travaux présentés dans cette thèse s'articulent autour de cet axe d'automatisation de l'analyse des données de supervision pour comprendre et prédire les défaillances. Nous avons proposé deux solutions principales mettant en œuvre des méthodes d'apprentissage supervisé permettant l'analyse automatisée des données de supervision.

Contributions

Pour analyser de manière automatisée des données de supervision afin de comprendre et de prédire les défaillances, nos travaux se sont articulés autour de deux axes principaux de recherche :

— La conception d'une solution permettant de prédire efficacement les surchauffes de processeur

— La conception d'une solution permettant de mettre en évidence les relations causales entre les événements survenant dans un système HPC à partir des données contenues dans les journaux systèmes.

Une évaluation pour valider les deux contributions est réalisée sur des données de supervision provenant d'un système HPC utilisé en production par DKRZ.

Prédiction des surchauffes des processeurs dans un système HPC

Nous proposons une solution utilisant un algorithme d'apprentissage machine pour la prédiction des surchauffes des processeurs dans les systèmes HPC.

La construction de la solution repose sur une analyse réalisée sur les conditions d'apparitions des événements de surchauffe. Cette analyse conclut que la seule information présente dans les données de supervision du système HPC étudié permettant de décrire et de prédire efficacement les surchauffes est la température du processeur. De plus, cette étude montre également que la surchauffe des processeurs peut augmenter significativement la durée des applications d'exécutant sur un processeur subissant une surchauffe.

La solution proposée, en utilisant les conclusions de l'analyse précédente, se repose sur l'analyse de la forme générale de la série temporelle représentant l'évolution de la température du processeur à travers le temps. Cette prise en compte de la forme nous permet de caractériser les différentes évolutions de la température des processeurs avant un événement de surchauffe. Ces caractéristiques sont ensuite utilisées par un

algorithme d'apprentissage machine pour découvrir de manière automatisée les relations entre l'évolution de la température et les événements de surchauffe.

Nous avons comparé cette approche d'analyse de la forme avec une approche se focalisant sur la prédiction des prochains points de température. Cette comparaison montre les bénéfices sur les résultats de notre approche qui permet d'analyser une tendance générale au lieu d'informations précises. De plus, l'analyse de la forme générale de la température permet de prédire plusieurs minutes en avance les surchauffes en utilisant des données de température collectées avec un faible taux d'échantillonnage.

Pour valider l'utilité globale de la solution, nous avons évalué la capacité de la solution à réduire le coût des surchauffes sur le système en utilisant une action préventive. Les deux actions préventives évaluées (une réduction anticipée de la fréquence et une migration de tâches) montrent que la solution proposée fournit des résultats aptes à réduire efficacement le coût des surchauffes en tenant compte à la fois des caractéristiques propres à chaque action (délai d'action, temps d'efficacité de l'action, etc.) mais également des caractéristiques des surchauffes (durée de la surchauffe, baisse associée de la fréquence du processeur, etc.).

Analyse des journaux systèmes

Nous proposons une solution utilisant une méthode d'apprentissage profond qui permet de fournir une vue augmentée des entrées des journaux système en mettant en évidence les relations causales entre elles.

Cette solution repose sur deux phases : la première phase est l'analyse des journaux systèmes pour la découverte des événements associés aux entrées et la deuxième phase est la découverte des relations causales entre les événements.

La première phase utilise un nouvel analyseur de journaux systèmes qui surpasse de plusieurs ordres de grandeur en temps d'exécution les différentes solutions évaluées provenant de l'état de l'art. Cette réduction du temps d'exécution est rendue possible grâce à l'utilisation d'une supposition forte mais viable sur la structure des journaux systèmes. Cette supposition permet d'obtenir un analyseur de journaux pouvant traiter les différentes entrées de manière parallèle. En complément de cette supposition, l'utilisation de représentations efficaces des entrées permet de réduire l'empreinte mémoire nécessaire. Ce temps d'exécution réduit permet l'utilisation en temps réel de cet analyseur pour analyser les journaux systèmes produits dans un système HPC. L'évaluation de la précision de cet analyseur montre qu'il est capable, malgré sa rapidité, d'atteindre les performances des analyseurs de l'état de l'art sur des jeux de données variées provenant de différents types de systèmes.

La deuxième phase utilise un algorithme d'apprentissage profond pour mettre en évidence les relations causales entre les événements précédemment détectés. L'utilisation d'un algorithme d'apprentissage profond est réalisée dans un contexte inhabituel. En effet, ces algorithmes sont généralement utilisés pour réaliser des prédictions. Nous proposons d'utiliser leurs capacités de prédiction pour être capable de détecter les relations causales entre les entrées en ajoutant une nouvelle couche à ces algorithmes. Cette nouvelle couche, appelée couche d'attention, s'appuie sur des travaux récents de l'état de l'art. Elle permet de déterminer l'importance des entrées de l'algorithme dans sa prédiction. En utilisant cette information, nous sommes capables de déduire les relations entre les entrées, représentants les événements précédents, et la sortie, représentant l'événement prédit.

L'évaluation de la solution sur le jeu de données provenant du système HPC de DKRZ, incluant les deux phases, montre qu'elle est capable de détecter les relations causales de la

grande majorité des événements survenant dans un système HPC. L'architecture de la solution permet de réaliser cette détection de manière totalement automatisée et avec un temps d'exécution compatible avec une utilisation dans un contexte interactif.

Perspectives

Les travaux présentés ouvrent des perspectives à plus ou moins long terme. Nous en présentons les principales en commençant par celles à court terme puis celles à plus long terme.

Analyse multi-nœuds

Dans l'état actuel, LogFlow n'est pas capable de découvrir les corrélations entre les journaux provenant des différents composants présents un système HPC, c'est-à-dire, qu'il n'est pas capable de trouver des liens de causalités entre les événements qui se produisent sur deux nœuds de calcul différents par exemple. Il se limite à découvrir les liens de causalités entre les entrées du journal d'un même nœud.

Pour découvrir des relations de causalité entre des événements se produisant dans différentes parties du système, nous pourrions utiliser la connaissance de l'architecture et des relations entre les différents composants d'un système HPC. Nous pourrions ainsi limiter notre recherche des relations causales aux composants connectés entre eux ou utilisés par la même application. En effet, il paraît peu probable qu'un événement sur un nœud donné est une relation causale avec un nœud physiquement à l'opposé du système et qui n'exécute pas la même application. Nous pouvons donc envisager d'utiliser directement la connaissance de l'architecture et des composants dans la solution pour ajouter de l'information et permettre une recherche plus efficace des relations causales.

Optimisation du refroidissement en utilisant le système de prédiction des surchauffes

Le système de refroidissement d'un système HPC est responsable d'une partie importante de sa consommation énergétique. La dernière décennie a vu l'émergence de solutions de refroidissement moins coûteuses que l'utilisation de climatisation comme par exemple l'utilisation d'eau tiède. Ces solutions ont pour but de réduire la consommation énergétique du système tout en garantissant son refroidissement et en évitant les surchauffes.

La solution de prédiction des surchauffes que nous avons proposée pourrait être utilisée pour optimiser la capacité de refroidissement en fonction des prédictions. En effet, il est envisageable de gérer la capacité de refroidissement dynamiquement : au minimum de sa capacité dans le cas courant et une augmentation progressive de la capacité en cas de prédiction d'une surchauffe. Cette variation de la capacité peut permettre de réduire la consommation énergétique en laissant un maximum de temps le système de refroidissement au minimum de sa capacité.

Optimisation des systèmes HPC grâce aux méthodes d'apprentissage machine

Dans cette thèse, nous avons étudié l'utilisation d'apprentissage machine comme moyen d'améliorer la gestion des défaillances. Cependant, il pourrait être aussi utilisé pour améliorer les performances du système HPC en utilisant les informations contenues dans

les données de supervision. Ces informations pourraient être utilisées pour optimiser plusieurs aspects d'un système HPC.

Par exemple, il serait possible d'optimiser les accès au système de fichier en analysant la façon dont les applications y accèdent. En effet, il est envisageable de catégoriser automatiquement les différents accès des applications à leurs données et de proposer des actions spécifiques pour permettre à chaque accès d'être le plus efficace possible. Par exemple, les données utilisées par l'application pourraient être pré-chargées avant leurs utilisations effectives car nous connaitrions à l'avance les données dont a besoin l'application en fonction de l'avancement de son exécution.

De la même manière, l'utilisation du réseau d'interconnexion peut être analysée pour proposer une configuration automatique efficace de l'application. Cette configuration peut se focaliser sur plusieurs aspects. Par exemple, il est possible de rapprocher automatiquement les nœuds d'une application qui communiquent le plus souvent entre eux ou d'éviter d'exécuter deux applications, qui utilisent de manière intensive le réseau, en même temps sur des nœuds proches.

Enfin, nous pourrions aussi imaginer de caractériser automatiquement l'exécution d'une application à l'aide de l'analyse d'un ensemble de métriques. Cette caractérisation permettrait par exemple au développeur de l'application de connaitre les goulets d'étranglement présents par rapport à l'utilisation sur un système particulier ou encore de savoir si le comportement de son application est similaire à celui d'une autre afin d'évaluer de manière plus précise l'efficacité de son application.

Bibliographie

[1] Aghabozorgi, S., Shirkhorshidi, A. S., and Wah, T. Y. Time-series clustering–a decade review. *Information Systems 53* (2015), 16–38.

[2] Agrawal, R., Srikant, R., et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB* (1994), vol. 1215, pp. 487–499.

[3] Alpaydin, E. *Introduction to machine learning*. MIT press, 2009.

[4] Aué, J. Log analysis from a to z : A literature survey. Master's thesis, Delft University of Technology, Delft, Netherlands, 2016.

[5] Austin, B., and Wright, N. J. Measurement and interpretation of microbenchmark and application energy use on the cray xc30. In *2014 Energy Efficient Supercomputing Workshop* (2014), IEEE, pp. 51–59.

[6] Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. The great time series classification bake off : a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery 31* (2017), 606–660.

[7] Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., and Yarrow, M. The nas parallel benchmarks 2.0. Tech. rep., Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[8] Bautista-Gomez, L., Gainaru, A., Perarnau, S., Tiwari, D., Gupta, S., Engelmann, C., Cappello, F., and Snir, M. Reducing waste in extreme scale systems through introspective analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2016), IEEE, pp. 212–221.

[9] Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N., and Matsuoka, S. Fti : high performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis* (2011), pp. 1–32.

[10] Bouguerra, M. S., Gainaru, A., Gomez, L. B., Cappello, F., Matsuoka, S., and Maruyam, N. Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing* (2013), IEEE, pp. 501–512.

[11] Bouteiller, A., Herault, T., Bosilca, G., and Dongarra, J. J. Correlated set coordination in fault tolerant message logging protocols. In *European Conference on Parallel Processing* (2011), Springer, pp. 51–64.

[12] Breiman, L. Random forests. *Machine learning 45* (2001), 5–32.

[13] Brown, A., Tuor, A., Hutchinson, B., and Nichols, N. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems* (2018), ACM, p. 1.

[14] Buda, M., Maki, A., and Mazurowski, M. A. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks 106* (2018), 249–259.

[15] Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B., and Snir, M. Toward exascale resilience. *The International Journal of High Performance Computing Applications 23*, 4 (2009), 374–388.

[16] Chen, Z., Fagg, G. E., Gabriel, E., Langou, J., Angskun, T., Bosilca, G., and Dongarra, J. Fault tolerant high performance computing by a coding approach. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming* (2005), ACM, pp. 213–223.

[17] Cho, J., Lee, K., Shin, E., Choy, G., and Do, S. How much data is needed to train a medical image deep learning system to achieve necessary high accuracy ? *arXiv preprint arXiv :1511.06348* (2015).

[18] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv :1406.1078* (2014).

[19] Council, J. Failure mechanisms and models for semiconductor devices. *JEDEC Publication JEP122-A* (2002).

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

[21] Das, A., Mueller, F., Hargrove, P., Roman, E., and Baden, S. Doomsday : Predicting which node will fail when on supercomputers. *Proceedings of the 31st International Conference on Supercomputing* (2018).

[22] Debnath, B., Solaimani, M., Gulzar, M. A. G., Arora, N., Lumezanu, C., Xu, J., Zong, B., Zhang, H., Jiang, G., and Khan, L. Loglens : A real-time log analysis system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (2018), IEEE, pp. 1052–1062.

[23] Di, S., Guo, H., Pershey, E., Snir, M., and Cappello, F. Characterizing and understanding hpc job failures over the 2k-day life of ibm bluegene/q system. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2019), IEEE, pp. 473–484.

[24] Di Martino, C., Kalbarczyk, Z., Iyer, R. K., Baccanico, F., Fullop, J., and Kramer, W. Lessons learned from the analysis of system failures at petascale : The case of blue waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2014), IEEE, pp. 610–621.

[25] Di Martino, C., Kramer, W., Kalbarczyk, Z., and Iyer, R. Measuring and understanding extreme-scale application resilience : A field study of 5,000,000 hpc application runs. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2015), IEEE, pp. 25–36.

[26] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. Querying and mining of time series data : experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment 1* (2008), 1542–1552.

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

[28] Du, M., Li, F., Zheng, G., and Srikumar, V. Deeplog : Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 1285–1298.

[29] El-Sayed, N., and Schroeder, B. Reading between the lines of failure logs : Understanding how hpc systems fail. In *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2013), IEEE, pp. 1–12.

[30] Elliott, J., Kharbas, K., Fiala, D., Mueller, F., Ferreira, K., and Engelmann, C. Combining partial redundancy and checkpointing for hpc. In *2012 IEEE 32nd International Conference on Distributed Computing Systems* (2012), IEEE, pp. 615–626.

[31] Elnozahy, E. N., Alvisi, L., Wang, Y.-M., and Johnson, D. B. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR) 34*, 3 (2002), 375–408.

[32] Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. Scalable object detection using deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2014), pp. 2147–2154.

[33] Ferreira, K., Stearley, J., Laros III, J. H., Oldfield, R., Pedretti, K., Brightwell, R., Riesen, R., Bridges, P. G., and Arnold, D. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (2011), pp. 1–12.

[34] Friedman, J., Hastie, T., and Tibshirani, R. *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.

[35] Friedman, J. H. Greedy function approximation : a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[36] Fu, Q., Lou, J.-G., Wang, Y., and Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining* (2009), IEEE, pp. 149–158.

[37] Gainaru, A., Cappello, F., Fullop, J., Trausan-Matu, S., and Kramer, W. Adaptive event prediction strategy with dynamic time window for large-scale hpc systems. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques* (2011), ACM, p. 4.

[38] Gainaru, A., Cappello, F., Snir, M., and Kramer, W. Fault prediction under the microscope : A closer look into hpc systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), IEEE Computer Society Press, p. 77.

[39] Gamboa, J. C. B. Deep learning for time-series analysis. *arXiv preprint arXiv :1701.01887* (2017).

[40] Gupta, S., Patel, T., Engelmann, C., and Tiwari, D. Failures in large scale systems : long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), ACM, p. 44.

[41] Hamooni, H., Debnath, B., Xu, J., Zhang, H., Jiang, G., and Mueen, A. Logmine : Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM*

International on Conference on Information and Knowledge Management (2016), ACM, pp. 1573–1582.

[42] Haque, A., Alahi, A., and Fei-Fei, L. Recurrent attention models for depth-based person identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 1229–1238.

[43] Hargrove, P. H., and Duell, J. C. Berkeley lab checkpoint/restart (blcr) for linux clusters. In *Journal of Physics : Conference Series* (2006), vol. 46, IOP Publishing, p. 494.

[44] He, P., Zhu, J., Xu, P., Zheng, Z., and Lyu, M. R. A directed acyclic graph approach to online log parsing. *arXiv preprint arXiv :1806.04356* (2018).

[45] He, P., Zhu, J., Zheng, Z., and Lyu, M. R. Drain : An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)* (2017), IEEE, pp. 33–40.

[46] Herault, T., and Robert, Y. *Fault-tolerance techniques for high-performance computing*. Springer, 2015.

[47] Hinton, G. E., et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society* (1986), vol. 1, Amherst, MA, p. 12.

[48] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[49] Htike, K. K., and Khalifa, O. O. Comparison of supervised and unsupervised learning classifiers for human posture recognition. In *International Conference on Computer and Communication Engineering (ICCCE'10)* (2010), IEEE, pp. 1–6.

[50] Jiang, Z. M., Hassan, A. E., Flora, P., and Hamann, G. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software* (2008), IEEE, pp. 181–186.

[51] Kim, M.-H., Ham, S.-W., Park, J.-S., and Jeong, J.-W. Impact of integrated hot water cooling and desiccant-assisted evaporative cooling systems on energy savings in a data center. *Energy 78* (2014), 384–396.

[52] Koutsikos, N. *Investigating power efficiency and co-location effects on heterogeneous HPC architectures*. PhD thesis, Master's thesis, 2013.

[53] Laskov, P., Düssel, P., Schäfer, C., and Rieck, K. Learning intrusion detection : supervised or unsupervised ? In *International Conference on Image Analysis and Processing* (2005), Springer, pp. 50–57.

[54] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[55] Lee, K., Booth, D., and Alam, P. A comparison of supervised and unsupervised neural networks in predicting bankruptcy of korean firms. *Expert Systems with Applications 29*, 1 (2005), 1–16.

[56] Liang, Y., Zhang, Y., Sivasubramaniam, A., Jette, M., and Sahoo, R. Bluegene/l failure analysis and prediction models. In *International Conference on Dependable Systems and Networks (DSN'06)* (2006), IEEE, pp. 425–434.

[57] Liang, Y., Zhang, Y., Sivasubramaniam, A., Sahoo, R. K., Moreira, J., and Gupta, M. Filtering failure logs for a bluegene/l prototype. In *2005*

International Conference on Dependable Systems and Networks (DSN'05) (2005), IEEE, pp. 476–485.

[58] Lively, C., Wu, X., Taylor, V., Moore, S., Chang, H.-C., and Cameron, K. Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems. *The International Journal of High Performance Computing Applications 25*, 3 (2011), 342–350.

[59] Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv :1508.04025* (2015).

[60] Makanju, A. A., Zincir-Heywood, A. N., and Milios, E. E. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009), ACM, pp. 1255–1264.

[61] Marathe, A., Bailey, P. E., Lowenthal, D. K., Rountree, B., Schulz, M., and de Supinski, B. R. A run-time system for power-constrained hpc applications. In *International conference on high performance computing* (2015), Springer, pp. 394–408.

[62] Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., et al. Loganomaly : Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization* (2019), vol. 7, pp. 4739–4745.

[63] Messaoudi, S., Panichella, A., Bianculli, D., Briand, L., and Sasnauskas, R. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension* (2018), ACM, pp. 167–177.

[64] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv :1301.3781* (2013).

[65] Mizutani, M. Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing* (2013), IEEE, pp. 595–602.

[66] Moody, A., Bronevetsky, G., Mohror, K., and De Supinski, B. R. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC'10 : Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (2010), IEEE, pp. 1–11.

[67] Nagappan, M., and Vouk, M. A. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (2010), IEEE, pp. 114–117.

[68] Nagarajan, A. B., Mueller, F., Engelmann, C., and Scott, S. L. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st annual international conference on Supercomputing* (2007), ACM, pp. 23–32.

[69] Netti, A., Kiziltan, Z., Babaoglu, O., Sîrbu, A., Bartolini, A., and Borghesi, A. Online fault classification in hpc systems through machine learning. In *European Conference on Parallel Processing* (2019), Springer, pp. 3–16.

[70] Nie, B., Xue, J., Gupta, S., Engelmann, C., Smirni, E., and Tiwari, D. Characterizing temperature, power, and soft-error behaviors in data center systems :

Insights, challenges, and opportunities. In *IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (2017), IEEE, pp. 22–31.

[71] Nie, B., Xue, J., Gupta, S., Patel, T., Engelmann, C., Smirni, E., and Tiwari, D. Machine learning models for gpu error prediction in a large scale hpc system. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2018), IEEE, pp. 95–106.

[72] Pei, W., Baltrusaitis, T., Tax, D. M., and Morency, L.-P. Temporal attention-gated model for robust sequence classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 6730–6739.

[73] Philp, I. Software failures and the road to a petaflop machine. In *HPCRI : 1st Workshop on High Performance Computing Reliability Issues, in Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)* (2005), pp. 125–128.

[74] Ropars, T., Guermouche, A., Uçar, B., Meneses, E., Kalé, L. V., and Cappello, F. On the use of cluster-based partial message logging to improve fault tolerance for mpi hpc applications. In *European Conference on Parallel Processing* (2011), Springer, pp. 567–578.

[75] Ropars, T., Martsinkevich, T. V., Guermouche, A., Schiper, A., and Cappello, F. Spbc : Leveraging the characteristics of mpi hpc applications for scalable checkpointing. In *SC'13 : Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2013), IEEE, pp. 1–12.

[76] Rountree, B., Ahn, D. H., De Supinski, B. R., Lowenthal, D. K., and Schulz, M. Beyond dvfs : A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum* (2012), IEEE, pp. 947–953.

[77] Rountree, B., Lownenthal, D. K., De Supinski, B. R., Schulz, M., Freeh, V. W., and Bletsch, T. Adagio : making dvs practical for complex hpc applications. In *Proceedings of the 23rd international conference on Supercomputing* (2009), pp. 460–469.

[78] Sawyer, R. Calculating total power requirements for data centers. *White Paper, American Power Conversion 562* (2004).

[79] Schroeder, B., and Gibson, G. A. Understanding failures in petascale computers. In *Journal of Physics : Conference Series* (2007), vol. 78, IOP Publishing, p. 012022.

[80] Shima, K. Length matters : Clustering system log messages using length of words. *arXiv preprint arXiv :1611.03213* (2016).

[81] Srinivasan, J., Adve, S. V., Bose, P., and Rivers, J. A. The impact of technology scaling on lifetime reliability. In *International Conference on Dependable Systems and Networks, 2004* (2004), IEEE, pp. 177–186.

[82] Stone, C. J. Classification and regression trees. *Wadsworth International Group 8* (1984), 452–456.

[83] Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 843–852.

[84] Tang, L., Li, T., and Perng, C.-S. Logsig : Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (2011), ACM, pp. 785–794.

[85] Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J., Leung, V. J., Egele, M., and Coskun, A. K. Online diagnosis of performance variation in hpc systems using machine learning. *IEEE Transactions on Parallel and Distributed Systems 30*, 4 (2018), 883–896.

[86] Turner, J., and Charniak, E. Supervised and unsupervised learning for sentence compression. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (2005), Association for Computational Linguistics, pp. 290–297.

[87] Vaarandi, R. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)* (2003), IEEE, pp. 119–126.

[88] Vaarandi, R., and Pihelgas, M. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International Conference on Network and Service Management (CNSM)* (2015), IEEE, pp. 1–7.

[89] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[90] Vinayavekhin, P., Chaudhury, S., Munawar, A., Agravante, D. J., De Magistris, G., Kimura, D., and Tachibana, R. Focusing on what is relevant : Time-series learning and understanding using attention. In *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), IEEE, pp. 2624–2629.

[91] Viswanath, R., Wakharkar, V., Watwe, A., Lebonheur, V., et al. Thermal performance challenges from silicon to systems.

[92] Wang, C., Mueller, F., Engelmann, C., and Scott, S. L. Hybrid checkpointing for mpi jobs in hpc environments. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems* (2010), IEEE, pp. 524–533.

[93] Wang, C., Mueller, F., Engelmann, C., and Scott, S. L. Proactive process-level live migration and back migration in hpc environments. *Journal of Parallel and Distributed Computing 72* (2012), 254–267.

[94] Wang, G., Zhang, L., and Xu, W. What can we learn from four years of data center hardware failures ? In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2017), IEEE, pp. 25–36.

[95] Wang, S., Liu, W., Wu, J., Cao, L., Meng, Q., and Kennedy, P. J. Training deep neural networks on imbalanced data sets. In *2016 international joint conference on neural networks (IJCNN)* (2016), IEEE, pp. 4368–4374.

[96] Wang, Z., Yan, W., and Oates, T. Time series classification from scratch with deep neural networks : A strong baseline. In *2017 international joint conference on neural networks (IJCNN)* (2017), IEEE, pp. 1578–1585.

[97] Whitle, P. *Hypothesis testing in time series analysis*, vol. 4. Almqvist & Wiksells, 1951.

[98] Wistuba, M., Rawat, A., and Pedapati, T. A survey on neural architecture search, 2019.

[99] Xing, Z., Pei, J., and Keogh, E. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter 12* (2010), 40–48.

[100] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell : Neural image caption generation with visual attention. In *International conference on machine learning* (2015), pp. 2048–2057.

[101] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 117–132.

[102] Yan, X., and Su, X. *Linear regression analysis : theory and computing*. World Scientific, 2009.

[103] Yun, B., Shin, K. G., and Wang, S. Predicting thermal behavior for temperature management in time-critical multicore systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th* (2013), IEEE, pp. 185–194.

[104] Zeiler, M. D., and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision* (2014), Springer, pp. 818–833.

[105] Zhang, M.-L., and Zhou, Z.-H. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering 26*, 8 (2013), 1819–1837.

[106] Zheng, Z., Lan, Z., Gupta, R., Coghlan, S., and Beckman, P. A practical failure prediction with location and lead time for blue gene/p. In *Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on* (2010), IEEE, pp. 15–22.

[107] Zheng, Z., Lan, Z., Park, B. H., and Geist, A. System log pre-processing to improve failure prediction. In *2009 IEEE/IFIP International conference on Dependable Systems & Networks* (2009), IEEE, pp. 572–577.

[108] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering : Software Engineering in Practice* (2019), IEEE Press, pp. 121–130.