Automatisation de Docker Swarm sur SoCs ARM avec support MPI et Analyse des Performances

Luiz Angelo Steffenel, Bruno da Silva Alves, Andrea Charão

Compas'2019 : Parallélisme/ Architecture / Système
LIUPPA - IUT de Bayonne, France, du 24 au 28 juin 2019

Automatisation de Docker Swarm sur SoCs ARM avec support MPI et Analyse des Performances

Luiz Angelo Steffenel[1], Bruno da Silva Alves[2] and Andrea Charao[2]

[1] Laboratoire CReSTIC, Université de Reims Champagne-Ardenne, Reims - France
luiz-angelo.steffenel@univ-reims.fr

[2] Laboratoire LSC, Universidade Federal de Santa Maria, Santa Maria, Brésil
bdalves@inf.ufsm.br, *andrea@inf.ufsm.br*

Résumé
La virtualisation par conteneurs représente une solution flexible et évolutive pour les environnements HPC, permettant une gestion simple et efficace des applications scientifiques. Récemment, les systèmes sur puce (*System-on-a-Chip* - SoC) se sont illustrés comme une alternative aux ordinateurs traditionnels, avec une bonne puissance de calcul et de faibles coûts. Dans cet article, nous décrivons le développement d'une solution basée sur des conteneurs pour les clusters SoC, et nous étudions la performance du WRF dans de tels environnements. Les résultats démontrent que même si les performances de pointe des SoC sont encore limitées, ces environnements sont plus qu'adaptés à certaines applications scientifiques.

Mots-clés : Conteneurs ; Système sur puce ; MPI ; ARM ; WRF

---

1. Introduction

Le calcul haute performance (HPC) est un terme générique désignant les applications à forte intensité de calcul ou de données de nature [16]. Alors que la plupart des plates-formes HPC reposent sur des infrastructures dédiées et coûteuses telles que les clusters et les grilles de calcul, d'autres technologies comme le cloud computing et les systèmes sur puce (*System-on-a-Chip* - SoC) représentent des alternatives intéressantes.
Les systèmes sur puce représentent une rupture avec les ordinateurs traditionnels car les SoC regroupent CPU, GPU, mémoire RAM et autres composants sur la même puce [18]. La plupart du temps, la technologie SoC est utilisée comme un moyen de réduire le coût des ordinateurs monocarte tels que les Raspberry Pi, ODroid ou Banana Pi. Ces systèmes sont actuellement utilisés pour un large éventail d'applications, de l'enseignement de l'informatique [1] à Internet des Objets [12]. Étant principalement basés sur des processeurs ARM, les SoCs bénéficient également des améliorations apportées à cette famille de processeurs, qui présente aujourd'hui des caractéristiques permettant la construction d'infrastructures HPC [17][7][13].
Dans ce contexte, l'association du SoC et de la virtualisation représente une solution intéressante pour déployer des applications scientifiques en production, en raison de leur faible coût et un besoin limité d'entretien et d'infrastructure. Afin d'évaluer l'intérêt de ces environnements, nous nous intéressons au déploiement d'un cluster SoC pour l'exécution de WRF [15], une application de modélisation météorologique qui peut être déployée sur un cluster grâce à

MPI. Au delà de l'intérêt en production, la virtualisation peut également favoriser un usage à des fins éducatives [2]. En effet, la virtualisation permet la création d'environnements expérimentaux pour les étudiants, qui peuvent tester différents logiciels sans avoir à se battre avec la maintenance et l'installation des applications.
Le reste de ce travail est structuré comme suit : la section 2 présente les obstacles au déploiement d'applications MPI sur un environnement Docker. La section 3 introduit le framework WRF et les adaptations que nous avons faites pour développer des images WRF Docker pour les SoCs. La section 4 présente quelques résultats d'analyse comparative obtenus et enfin la section 5 présente les conclusions tirées de cette étude et nos plans pour les travaux futurs.

2. Déploiement de MPI sur Docker

Lorsqu'on envisage des applications HPC à grande échelle, souvent on s'appuie sur MPI pour l'échange de données et la coordination des tâches. Malgré les avancées récentes dans sa spécification, le déploiement d'une application MPI peut être assez rigide car il nécessite un environnement d'exécution bien connu. En effet, le point de départ d'un cluster MPI est la définition d'une liste de nœuds participants (souvent appelée `hostfile`), ce qui impose une connaissance préalable de l'environnement d'exécution.
Le déploiement d'un cluster MPI sur des conteneurs Docker est souvent une tâche difficile car le réseau d'interconnexion interne (overlay) est conçu pour l'équilibrage de charge et non pour adresser des nœuds spécifiques, comme dans le cas de MPI. La plupart des travaux qui proposent des images Docker pour MPI ne parviennent pas à développer une solution autonome, nécessitant une manipulation manuelle ou externe des éléments MPI pour déployer les applications. Dans le cas de [9], par exemple, l'orchestration des conteneurs est remplacée par une combinaison d'un gestionnaire de ressources (PBS) et d'un ensemble de scripts qui déploient des images individuelles puis les interconnectent. Une approche similaire est utilisée par [3], qui utilise Slurm comme orchestrateur.
Un autre exemple de "gestion externe" est le cas de Singularity [10], un gestionnaire de conteneurs spécialement conçu pour la communauté HPC. Singularity a développé une solution spécifique pour les déploiements MPI, où un outil externe déploie et configure le *hostfile* MPI. Dans [5], MPI ne fait même pas partie du conteneur mais est monté depuis l'OS hôte, rendant leur solution totalement dépendante de la plate-forme d'exécution.
Enfin, [14], propose un service générique pour le déploiement d'applications MPI sur Docker en une machine ou en cluster, avec Docker Swarm. Basée sur la distribution Alpine Linux, cette plate-forme automatise la plupart du déploiement du service Docker Swarm, obtenant la liste des nœuds de travail grâce à la surveillance des connexions actives avec `netstat`. Le choix d'utiliser `netstat` s'avère cependant trop instable, et nous n'avons pas pu le faire fonctionner correctement sur un SoC.
Comme les solutions existantes nécessitent trop d'interventions manuelles ou ne sont pas fiables, nous avons décidé de développer notre propre solution pour automatiser le déploiement du MPI sur un cluster Docker Swarm en SoC, comme suit.

2.1. Création automatique du hostfile

La principale difficulté ici est due au fait que Docker présente deux modes d'exécution assez différents : dans le mode "standalone", une instance de conteneur est lancée comme une application autonome, ne nécessitant aucune interconnexion à d'autres instances (bien que cela soit possible). En mode "service", différentes instances sont reliées par un réseau *overlay*. Dans les deux cas, il n'y a pas une façon simple de dresser une liste de tous les nœuds en exécution.

Nous avons ainsi besoin d'un service de découverte pour identifier quelles adresses IP correspondent aux instances de notre réseau. Contrairement à [14], nous avons utilisé le service de nommage de Docker en faisant des appels DNS "bas niveau" avec *dig*. Grâce à ces requêtes, on obtient une liste des adresses IP correspondantes aux instances de notre application. Comme le *hostfile* indique également le nombre de processus MPI qu'un noeud peut exécuter simultanément, on fail appel à l'application *nproc* sur chaque machine, obtenant ainsi le nombre de cœurs disponibles. Ce simple "hack" est présenté ci-dessous, où l'on obtient la liste de tous les nœuds *worker* (c'est-à-dire les instances du service "worker") sur Swarm.

```
iplists=`dig +short tasks.workers A`
for i in $iplists; do
np=`ssh $i "nproc --all"`
echo "$i:$np" >> hostfile
done
```

2.2. Service, rôles et accès externe

En plus de la liste des nœuds, MPI s'appuie fortement sur leur *rang*. Comme le nœud "maître" (rang 0) est souvent utilisé pour lancer les applications et récupérer les résultats, il est important de permettre l'accès à ce nœud via SSH. Par conséquent, nous avons créé un service Docker Swarm qui différencie l'instance maître des autres, permettant ainsi la publication du port du service du maître ainsi que le montage des volumes externes nécessaires à l'application.

3. Le Modèle WRF

Afin d'expérimenter notre plateforme virtualisée, nous avons choisi d'exécuter la suite WRF (Weather Research and Forecasting) [15], un modèle de prévision météorologique numérique très connu. WRF a plus de 1,5 million de lignes en C et Fortran, ainsi que de nombreuses dépendances à l'égard de bibliothèques externes pour les entrées/sorties, les communications parallèles et la compression de données. Par conséquent, la compilation et l'exécution peuvent être difficiles pour les débutants ou pour les utilisateurs qui n'ont pas les droits d'administration.

Le workflow typique pour exécuter le modèle WRF est composé par 5 étapes, sans compter l'accès supplémentaire à des sources de données externes, ni l'analyse/visualisation des résultats :

1. Geogrid - crée des données terrestres à partir de données géographiques statiques
2. Ungrib - décompresse les données météorologiques GRIB obtenues d'une source externe et les compresse dans un format de fichier intermédiaire.
3. Metgrid - interpolation horizontale des données météorologiques sur le domaine du modèle.
4. Real - interpolation verticale des données sur les coordonnées du modèle, création de fichiers de conditions limites et initiales, et contrôles de cohérence.
5. WRF - génère la prévision du modèle

Les trois premières étapes font partie du système de prétraitement WPS, qui est configuré et compilé séparément du modèle WRF. La configuration WPS permet deux modes d'exécution : `serial` or `dmpar` (parallélisme de la mémoire distribuée par MPI).

Dans le cas de la configuration du modèle WRF, quatre modes sont proposés : `serial`, `smpar` (parallélisme de mémoire partagée), `dmpar` (parallélisme de mémoire distribuée) et `sm+dmpar`. L'option `smpar` dépend d'OpenMP, tandis que l'option `dmpar` repose sur MPI. La dernière

option (`sm+dmpar`) combine OpenMP et MPI, mais plusieurs travaux soulignent que `dmpar` surpasse généralement l'option mixte [6][11].
Compte tenu des nombreuses dépendances et des multiples options de configuration, l'installation de WRF est souvent laborieuse. Grâce aux conteneurs, nous pouvons offrir un moyen de simplifier le déploiement d'infrastructures informatiques pour l'enseignement et la recherche.

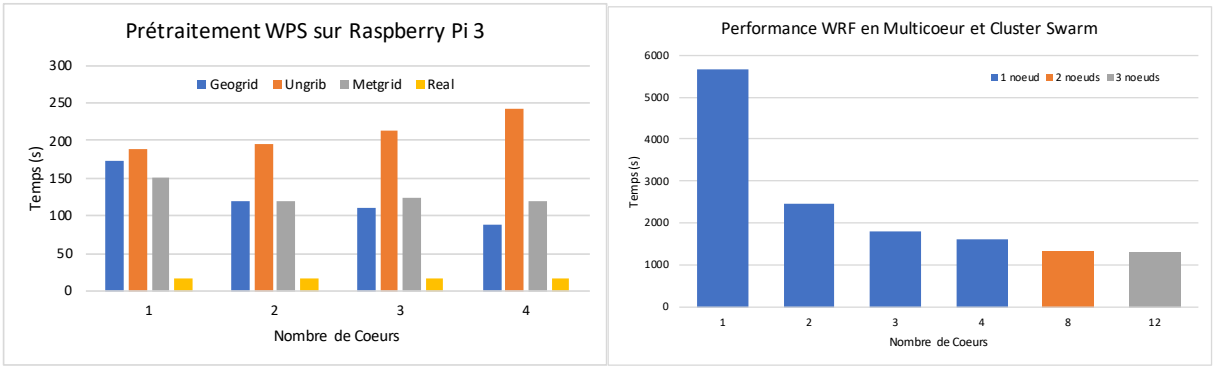
Compilation de WRF sur ARM
Bien qu'il existe déjà un conteneur non officiel pour le WRF sur les plates-formes x86 [8], cette image n'est pas adaptée au déploiement en cluster et n'a pas évolué depuis son lancement. Lorsque nous avons commencé à développer une version compatible avec les SoCs, nous avons dû résoudre des problèmes liés à la compatibilité avec l'architecture ARM, ainsi que la disponibilité de certaines bibliothèques. Pour cette raison, nous avons dû migrer l'image de base du conteneur vers `ubuntu`, qui non seulement supporte ARM comme propose la plupart des bibliothèques requises par WRF sous forme de paquets, simplifiant l'installation.
De plus, nous avons modifié la façon d'accéder aux données d'entrée, en passant d'un volume fixe de Docker à un système de fichiers monté. Ceci donne plus de flexibilité aux utilisateurs, qui peuvent manipuler les données indépendamment des volumes Docker, et permet également de résoudre un problème de stockage sur les SoC. En effet, la première étape du workflow WRF (`Geogrid`) dépend d'une large base de données géographique (`WPS_GEOG`) qui atteint 60 Go. En permettant l'utilisation de volumes externes, nous donnons aux utilisateurs la possibilité d'attacher des disques de stockage externes à leurs nœuds. Comme démontré en section 4.2, un seul nœud maître nécessite cette base de données, ce qui permet de minimiser les coûts et la complexité de gestion du cluster SoC.
Les conteneurs WRF pour les architectures ARM et x86 sont disponibles sur Docker Hub [1] et les scripts et les fichiers Docker sont disponibles sur github [2].

4. Évaluation des Performances

Afin d'évaluer l'intérêt et les limitations d'un déploiement sur un cluster SoC basé sur ARM pour les simulations avec WRF, nous avons mené une série de benchmarks. Les sessions suivantes décrivent les expériences et les plateformes que nous avons comparées.

4.1. Description de l'environnement d'exécution
Pour les tests de performance, nous avons utilisé un dataset permettant la simulation de 12h de prévisions météorologiques sur une zone couvrant l'Uruguay et le sud du Brésil. Bien que petit, cet ensemble de données est souvent utilisé par les étudiants en météorologie à l'UFSM, qui s'entraînent à modifier les paramètres et comparer les résultats aux observations réelles. L'ensemble des données est accessible dans notre dépôt github.
Dans les benchmarks, nous avons comparé différents modèles de SoC, dont des **Raspberry Pi 2 modèle B** (Broadcom BCM2835, ARM Cortex-A7, 4 cœurs, 900MHz, 1GB RAM) et des **Raspberry Pi 3** (Broadcom BCM2837, ARM Cortex-A53, 4 cœurs, 1.2GHz, 1GB RAM). Nous avons également expérimenté avec d'autres cartes SoC comme NanoPi NEO (Allwinner H3, ARM Cortex-A7, 4 cœurs, 1.2GHz, 512MB RAM), un NTC C.H.I.P. (AllWinner R8, ARM Cortex-A8, 1 cœur, 1GHz, 512MB RAM) et un Banana Pi (Allwinner A83T, Arm Cortex-A7 8 cœurs, 1.8 GHz, 1 Go de RAM), mais leurs mauvaises performances ou incompatibilités avec Docker nous ont contraints à exclure ces plates-formes des tests ici présentés.

1. `https://hub.docker.com/r/lsteffenel/wrf-container-armv7l/`
2. `https://github.com/lsteffenel/wrf-container-armv7l-RaspberryPi`



(a) Impact du multicœur sur le prétraitement (b) Simulations WRF en multicœur et multinœud

FIGURE 1 – Performances des étapes de prétraitement et de simulation

Toutes les mesures présentées dans cette section correspondent à la moyenne d'au moins 5 exécutions. Pour les clusters Docker Swarm, nous avons relié les appareils via un commutateur 1 Gbps sur RJ45, pour éviter des résultats peu fiables dus aux connexions sans fil.
De plus, le workflow WRF étant composé de 5 étapes, nous avons calculé le temps d'exécution de chaque étape individuellement, afin d'évaluer la meilleure stratégie de déploiement. Par conséquent, les sections suivantes présentent l'analyse séparée des étapes de prétraitement (WPS+Real) et de l'étape de prévision (WRF).

4.2. Prétraitement avec WPS et Real

Comme expliqué dans la section 3, le stockage de la base de données géographique utilisée par l'étape Geogrid sur WPS (environ 60Go) pose un problème à cause de sa taille. En effet, les équipements utilisés dans nos expériences utilisent un stockage en mémoire SD, dont le prix devient trop important par rapport au prix du SoC s'il faut accommoder cette quantité de données. Ainsi, dans nos expériences, nous avons opté par attacher un périphérique de stockage USB externe au nœud maître.
Puisque WPS peut être compilé avec l'option `dmpar`, nous avons d'abord essayé d'identifier si l'utilisation de MPI serait bénéfique à chacune des étapes de WPS. Le résultat de ce benchmark, illustré dans la Figure 1a, indique que seule l'étape *Geogrid* bénéficie d'une exécution multicœur. Dans le cas de *Ungrib*, l'exécution parallèle pénalise même l'algorithme. L'étape *Metgrid* montre un petit gain de performance lors de la parallélisation mais le temps d'exécution se stabilise pour 2 noyaux ou plus, et l'étape *Real* ne montre aucune amélioration.
Même si *Geogrid* présente quelques améliorations de performance en parallèle, l'accélération reste sous-optimale (50% de gain pour 4x plus de ressources). Associé aux limitations de stockage citées précédemment et à son impact relativement faible sur le temps d'exécution global (en comparant avec l'étape de prévision, voir Section 4.3), nous déconseillons d'exécuter *Geogrid* sur tout le cluster. À partir de ces résultats, nous suggérons plutôt d'assigner un seul nœud (le `master`) pour les tâches de prétraitement, et d'utiliser l'exécution multicœur uniquement dans le cas de *Geogrid* et *Metgrid*.

4.3. Simulation avec WRF

Contrairement aux étapes de prétraitement qui ne représentent finalement qu'une faible charge de calcul, l'étape de prévision WRF a un besoin en calcul bien plus importante. Ceci est encore plus évident dans un environnement production, où les prévisions couvrent plusieurs jours.
Ainsi, l'utilisation de l'exécution multicœur sur un cluster peut bénéficier l'étape WRF. La figure 1b indique le temps d'exécution moyen lors de l'exécution de l'étape WRF sur un seul

Raspberry Pi 3 (1 à 4 noyaux), sur un cluster avec deux Raspberry Pi 3 (totalisant 8 noyaux) et sur un cluster avec 3 Raspberry Pi (deux RPi 3 et un RPi 2 totalisant 12 noyaux).
Si l'exécution multicœur permet un gain de performance important (71% d'accélération en passant de 1 à 4 cœurs), les exécutions Swarm Cluster montrent des résultats plus mitigés, avec une accélération peu importante (17% d'accélération entre 1 et 2 machines, et seulement 2% entre 2 et 3 machines). Bien que la performance de l'Overlay Docker mérite une investigation plus poussée, à l'instar de la comparaison effectuée par [19], nous soupçonnons aussi des problèmes de performance liés à l'interface réseau des Raspberry Pis. En effet, comme l'observe [4], l'accès au bus de communication est un problème récurrent sur les SoC, et les Raspberry Pi ne disposent que d'une carte d'interconnexion à "basse vitesse" (10/100 Mbps uniquement). De ce fait, la répartition des tâches sur un cluster serait pénalisée par la communication réseau.
Le Tableau 1 détaille ces résultats. L'apport de l'exécution multicœur et du multinœud permet un gain de performance qui, malgré une accélération non linéaire, permet la production des résultats dans un délai acceptable sur des SoC Raspberry Pi, ce qui est suffisant pour fournir des prévisions sur une base quotidienne ou même horaire. Si l'on considère le coût matériel et environnemental de la solution SoC, c'est en effet une alternative intéressante pour des applications scientifiques comme le WRF.

TABLE 1 – Performance d'execution de WRF en multicœur et sur un cluster

Cœurs	R Pi 2	R Pi 3	2 x R Pi 3	2 x R Pi 3 + R Pi 2
1	6268.96	5647.47	-	-
2	3280.34	2473.89	-	-
3	2468.89	1801.18	-	-
4	2075.88	1602.68	-	-
8	-	-	1322.42	-
12	-	-	-	1306.10

5. Conclusions

Ce travail se concentre sur le déploiement d'applications scientifiques conteneurisées sur un cluster de systèmes sur puce (SoC). La plupart des SoC sont basés sur l'architecture ARM, une famille de processeurs qui a commencé à pénétrer le domaine du calcul haute performance. La virtualisation à base de conteneurs, d'autre part, permet l'empaquetage d'applications complexes et les déployer en toute transparence. Ensemble, SoC et conteneurs représentent une alternative prometteuse pour le développement d'infrastructures informatiques, associant le faible coût et la maintenance minimale des SoCs avec la flexibilité des conteneurs.
Cependant, plusieurs applications scientifiques s'appuient sur MPI, et les gestionnaires de conteneurs populaires comme Docker n'offrent pas un support approprié pour MPI. Nous proposons donc des outils pour automatiser le déploiement d'un cluster Docker Swarm supportant les applications MPI. Afin d'évaluer la performance et l'intérêt des conteneurs sur des clusters SoC, nous avons étudié les performances du modèle météorologique WRF.
Les améliorations futures comprennent le développement d'une plate-forme générique capable d'accueillir d'autres applications MPI, ainsi que le développement d'un framework pour la mise en production de WRF sur un cluster SoC afin de générer des prévisions journalières.

Remerciements

Ce travail a été partiellement financé par le projet franco-brésilien CAPES-COFECUB MESO[3] et le projet GREEN-CLOUD[4] (#16/2551-0000 488-9 - FAPERGS et CNPq Brésil, programme PRONEX 12/2014).

Bibliographie

1. Ali (M.), Vlaskamp (J. H. A.), Eddin (N. N.), Falconer (B.) et Oram (C.). – Technical development and socioeconomic implications of the raspberry pi as a learning tool in developing countries. – In *Computer Science and Electronic Engineering Conf. (CEEC)*, pp. 103–108. IEEE, 2013.
2. Alvarez (L.), Ayguade (E.) et Mantovani (F.). – Teaching hpc systems and parallel programming with small-scale clusters. – In *2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, pp. 1–10, Nov 2018.
3. Azab (A.). – Enabling docker containers for high-performance and many-task computing. – In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 279–285, April 2017.
4. Beserra (D.), Pinheiro (M. K.), Souveyet (C.), Steffenel (L. A.) et Moreno (E. D.). – Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems. – In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 363–370, March 2017.
5. Chung (M. T.), Quang-Hung (N.), Nguyen (M.) et Thoai (N.). – Using docker in high performance computing applications. – In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pp. 52–57, July 2016.
6. Council (H. A.). – *Weather Research and Forecasting (WRF) : Performance Benchmark and Profiling, Best Practices of the HPC Advisory Council*. – Rapport technique, http ://www.hpcadvisorycouncil.com/pdf/ WRF_Analysis_and_Profiling_Intel.pdf, HPC Advisory Council, 2010.
7. Cox (S. J.), Cox (J. T.), Boardman (R. P.), Johnston (S. J.), Scott (M.) et O'brien (N. S.). – Iridis-pi : a low-cost, compact demonstration cluster. *Cluster Computing*, vol. 17, n2, 2014, pp. 349–358.
8. Hacker (J. P.), Exby (J.), Gill (D.), Jimenez (I.), Maltzahn (C.), See (T.), Mullendore (G.) et Fossell (K.). – A containerized mesoscale model and analysis toolkit to accelerate classroom learning, collaborative research, and uncertainty quantification. *Bulletin of the American Meteorological Society*, vol. 98, n6, 2017, pp. 1129–1138.
9. Higgins (J.), Holmes (V.) et Venters (C.). – Orchestrating docker containers in the hpc environment. – In Kunkel (J. M.) et Ludwig (T.) (édité par), *High Performance Computing*, pp. 506–513, Cham, 2015. Springer International Publishing.
10. Kurtzer (G. M.), Sochat (V.) et Bauer (M. W.). – Singularity : Scientific containers for mobility of compute. *PLOS ONE*, vol. 12, n5, 05 2017, pp. 1–20.
11. Langkamp (T.) et Böhner (J.). – Influence of the compiler on multi-cpu performance of wrfv3. *Geoscientific Model Development*, vol. 4, n3, 2011, pp. 611–623.
12. Molano (J. I. R.), Betancourt (D.) et Gómez (G.). – Internet of things : A prototype architecture using a raspberry pi. *In : Knowledge Management in Organizations*, pp. 618–631. – Springer, 2015.
13. Montella (R.), Giunta (G.) et Laccetti (G.). – Virtualizing high-end gpgpus on arm clusters

3. http://meso.univ-reims.fr
4. http://www.inf.ufrgs.br/greencloud/

for the next generation of high performance cloud computing. *Cluster computing*, vol. 17, n 1, 2014, pp. 139–152.

14. Nguyen (N.) et Bein (D.). – Distributed MPI cluster with Docker Swarm mode. – In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–7, Jan 2017.
15. Skamarock (W. C.), Klemp (J. B.), Dudhia (J.), Gill (D. O.), Barker (D. M.), Duda (M. G.), Huang (X.-Y.), Wang (W.) et Powers (J. G.). – A description of the advanced research wrf version 3, ncar technical note. *National Center for Atmospheric Research, Boulder, Colorado, USA*, 2008.
16. Somasundaram (T. S.) et Govindarajan (K.). – Cloudrb : A framework for scheduling and managing high-performance computing (hpc) applications in science cloud. *Future Generation Computer Systems*, vol. 34, 2014, pp. 47–65.
17. Weloli (J. W.), Bilavarn (S.), Vries (M. D.), Derradji (S.) et Belleudy (C.). – Efficiency modeling and exploration of 64-bit arm compute nodes for exascale. *Microprocessors and Microsystems*, vol. 53, 2017, pp. 68 – 80.
18. Wolf (W.), Jerraya (A. A.) et Martin (G.). – Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, n 10, 2008, pp. 1701–1713.
19. Yong (C.), Lee (G.-W.) et Huh (E.-N.). – Proposal of container-based hpc structures and performance analysis. vol. 14, n6, 2018.