**RENPAR'16 / CFSE'4 / SympAAA'2005 / Journées Composants**
**Le Croisic, France, 5 au 8 avril 2005**

# ARTiS, un système d'exploitation temps-réel asymétrique*

Philippe MARQUET, Philippe.Marquet@lifl.fr
Éric PIEL, Eric.Piel@lifl.fr
Julien SOULA, Julien.Soula@lifl.fr
Jean-Luc DEKEYSER, Jean-Luc.Dekeyser@lifl.fr

Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille
France

**Résumé**
Le système ARTiS est une extension temps-réel de GNU/Linux dédiée aux architectures multiprocesseurs symétriques (SMP). ARTiS exploite la caractéristique SMP de l'architecture pour garantir la possible préemption d'un processeur quand le système doit ordonnancer une tâche temps-réel. Le principe d'ARTiS est d'identifier un ensemble de processeurs dédiés aux opérations temps-réel. Un mécanisme de migration automatique des activités non préemptibles assure une garantie de latence sur ces processeurs temps-réel. De plus, une stratégie spécifique d'équilibrage de charge permet à ARTiS d'exploiter la pleine puissance d'une machine SMP : les réservations temps-réel, bien que garanties, ne sont pas exclusives et n'entraînent pas de sous-utilisations des ressources. Des simulations du comportement d'ARTiS ont permis de vérifier la viabilité du modèle proposé. Nous présentons ici une évaluation des performances, en particulier en terme de latence de traitement des interruptions, de différentes versions du noyau Linux et de notre solution ARTiS. Notre première implantation sur Intel x86 et IA-64, bien qu'incomplète, confirme la supériorité de la solution ARTiS sur le noyau Linux standard.

**Mots-clés :** Linux, temps-réel, ordonnancement, multiprocesseur, SMP, asymétrique

## 1. Linux, SMP et le temps-réel

Les domaines d'applications nécessitant un support temps-réel dur du système d'exploitation sont nombreux : des tâches applicatives peuvent nécessiter des communications avec un matériel dédié en utilisant un protocole temporellement contraint, par exemple pour assurer une acquisition temps-réel. Ces mêmes applications temps-réel peuvent aussi nécessiter une puissance de calcul importante, par exemple pour traiter de multiples flux vidéo. L'utilisation de systèmes SMP (Symmetric Multi-Processors) peut fournir cette puissance de calcul.

Le marché des systèmes d'exploitation temps-réel regorge de solutions propriétaires. Malgré la définition d'une interface POSIX standard pour les applications temps-réel [8], de nombreux vendeurs proposent leur propre API dédiée. Cette situation peut être la conséquence de l'absence d'un acteur majeur dans la communauté temps-réel, et nous sommes persuadés du succès d'une proposition d'un système d'exploitation temps-réel Open Source.

Notre objectif est ainsi d'identifier un système d'exploitation Open Source pour le temps-réel pour systèmes SMP :

- l'utilisation d'un système Open Source peut garantir une conformité avec les standards ;
- un système d'exploitation « complet » permet la cohabitation d'activités temps-réel avec les activités généralistes habituelles d'un système d'exploitation ;
- le support des architectures SMP permet de répondre aux besoins calculatoires des applications visées.

* Ce travail est partiellement supporté par le projet ITEA 01010, HYADES

Nous avons identifié quatre catégories de systèmes d'exploitation à même de satisfaire les besoins requis. Ces catégories sont :
– les systèmes d'exploitation temps-réel dédiés tels VxWorks ;
– le système GNU/Linux, et plus particulièrement les dernières versions du noyau dites « préemptibles » ;
– les extensions existantes de cohabitation du noyau Linux avec un micro-noyau telles RTLinux ou RTAI ;
– les systèmes d'exploitation basés sur l'approche AMP (Asymmetric Multi-Processing).
Cependant, aucune de ces solutions n'est pleinement satisfaisante au regard de notre cahier des charges.

**Systèmes d'exploitation temps-réel dédiés**

Nombre de systèmes d'exploitation temps-réel dédiés sont disponibles. Ils sont robustes et assurent de manière ferme des garanties temps-réel dures. Cependant, les cibles initiales de ces systèmes étant les applications embarquées, les mécanismes utiles à de grosses applications ne sont pas toujours présents. Le fait qu'un système tel VxWorks n'ait longtemps pas envisagé de supporter un mécanisme de protection mémoire est rédhibitoire [4]. De plus, le support des architectures SMP est souvent réduit à une configuration « multi-monoprocesseur », une instance du système d'exploitation s'exécutant sur chacun des processeurs. Cette approche complique la programmation d'applications désirant se déployer sur l'ensemble des processeurs d'une architecture SMP et limite le passage à l'échelle, avantage majeur des architectures SMP.

**Le système GNU/Linux**

Le système GNU/Linux est un système d'exploitation Open Source. Son exploitation des architectures SMP est maintenant mature [3], ses performances non temps-réel excellentes. Cependant, la construction même du noyau Linux contraint les garanties de latence souhaitées, que ce soit au niveau du noyau ou au niveau utilisateur. Le noyau Linux n'est pas préemptible et l'exécution du traitement associé à une interruption est en partie reportée à la fin d'un appel système en cours.
Les propositions d'amélioration de la latence du noyau Linux sont nombreuses. Le vendeur de solution Linux embarquées MontaVista est à l'origine d'une amélioration simple et systématique du noyau Linux [13] qui met en évidence des points de préemption dans le noyau, réduisant ainsi la latence du noyau. Ce patch, maintenu par Robert Love, a été adopté dans la branche principale du noyau [14], principalement parce qu'il amène la réduction de latence attendues par les applications multimédias.
Une approche complémentaire d'Ingo Molnar et Andrew Morton, nommée « low-latency » [19], ajoute des points de préemption fixes dans le noyau. Si les latences du noyau s'en trouvent réduites, la maintenance de ce patch avec l'évolution constante du noyau est une charge délicate et lourde. L'amélioration de la latence noyau reste alors du ressort d'experts dans le domaine.
Plus récemment, Ingo Molnar a proposé une approche similaire mais plus facilement maintenable, la « voluntary kernel preemption », qui améliore encore la latence du noyau. Le principe est qu'un appel à la fonction `might_sleep()`, jusqu'à présent utilisée uniquement pour la vérification des spinlocks, puisse entraîner un appel à l'ordonnanceur. De son côté, Sven-Thorsten Dietrich de MontaVista a annoncé un ensemble de modifications indépendantes améliorant la latence du noyau. Parmi ces propositions, on note une implantation de spinlocks par des mutex et la gestion d'IRQ par des fils d'exécution du noyau. Ces travaux sont tous deux en cours et visent principalement le multimédia temps-réel.
Une alternative, proposée par Bernard Kuhn sous la forme d'interruptions temps-réel introduisant une notion de priorité entre les interruptions [9], permet de diminuer la latence dans le pire cas. Cependant, l'extension de ce mécanisme à des architectures SMP n'est pas immédiat, une interruption prioritaire pouvant être retardée parce qu'elle partage un verrou avec l'exécution, sur un autre processeur, d'une interruption moins prioritaire.

**Approche co-noyau**

L'approche dite co-noyau se vante d'ajouter des capacités temps-réel au noyau Linux. Ces extensions à Linux consistent en un petit noyau temps-réel fournissant les services temps-réel et qui tourne le noyau Linux comme une tâche de priorité faible quand aucune tâche temps-réel n'est éligible. Les interruptions

matérielles sont reroutées au noyau Linux par le noyau temps-réel. Cette virtualisation des interruptions du noyau Linux permet au co-noyau de préempter le noyau Linux quand nécessaire. RTLinux [7, 20] et RTAI [5] sont deux représentants fameux des systèmes basés sur ce principe.
Si les versions récentes de RTLinux visent aussi les architectures SMP [21], RTLinux est couvert par une licence « Open RTLinux Patent License » ou par une licence commerciale et utilise un brevet des FSM Labs qui peut en freiner l'usage ou l'adoption, en dépit de son succès actuel.
Cette approche co-noyau n'unifie pas les systèmes Linux et temps-réel, mais offre une double plateforme au développeur : les processus temps-réel ne bénéficient pas des services du noyau Linux, alors que les processus Linux ne peuvent profiter des services temps-réel. C'est un inconvénient majeur, même si RTLinux supporte des communications entre les processus temps-réel et les processus Linux au travers de FIFO temps-réel [6], ou si RTAI fournit une API unique au travers un module noyau nommé LXRT qui exporte les services temps-réel aux processus Linux.

**Multiprocessing asymétrique**

Une dernière approche, nommée multiprocessing asymétrique, exploite l'architecture SMP en introduisant une notion de processeurs protégés (*shielded*). Sur une machine multiprocesseurs, les processeurs sont spécialisés, temps-réel ou non : les processeurs temps-réel vont exécuter les tâches temps-réel alors que les processeurs non temps-réel vont exécuter les tâches non temps-réel. La modification RedHawk Linux de Concurrent Computer Corporation [2, 1] et le système REACT/pro de SGI [17], une extension de IRIX, suivent ce principe.
Cependant, seules les tâches temps-réel étant autorisées à s'exécuter sur les processeurs protégés, si ces tâches n'en consomment pas toute la puissance, de la ressource processeur est gâchée. Nous avions cependant validé la faisabilité de cette approche dans des travaux antérieurs [12]. Notre proposition d'ARTiS améliore cette notion élémentaire d'ordonnancement temps-réel asymétrique en autorisant le partage des ressources entre les tâches temps-réel et les tâches non temps-réel.

**Contenu de la contribution**

La section suivante contient une description de notre proposition ARTiS, modification du noyau Linux pour le temps-réel, et énonce les concepts proposés à l'utilisateur. La section 3 illustre par un exemple une utilisation typique des concepts proposés. La section suivante détaille un protocole d'évaluation de la latence noyau et compare les performances obtenues sur différentes configurations des noyaux Linux et ARTiS. La section 5 fournit les grandes lignes de l'implantation d'ARTiS. Un lecteur intéressé par plus de détails sur ce dernier point se référera à [11].

## 2. ARTiS : un ordonnanceur temps-réel asymétrique

Notre proposition est une contribution à la définition d'une extension temps-réel de Linux pour les machines SMP. Le modèle de programmation fourni par ARTiS repose sur la définition au niveau utilisateur de tâches temps-réel : le programmeur utilise les interfaces POSIX et/ou Linux pour définir son application. Ces tâches sont temps-réel en ce sens qu'elles sont associées à une priorité maximale et qu'elles ne seront perturbées par aucune activité non temps-réel. Typiquement, pour ces tâches, nous envisageons des temps de réponse inférieurs à 300$\mu$s. Cette borne a été fixée après une étude des besoins des partenaires industriels du projet.
Afin de bénéficier des architectures SMP, un système d'exploitation doit prendre en compte le mécanisme de mémoire partagée, la migration et l'équilibrage de charge entre les processeurs, et les schémas de communications entre les tâches. La complexité d'un tel système d'exploitation l'apparente plus aux systèmes d'exploitation généralistes qu'aux systèmes propriétaires temps-réel. Un système d'exploitation temps-réel sur machine SMP doit implanter l'ensemble de ces mécanismes et considérer leur interaction avec des contraintes temporelles dures. Un tel cahier des charges participe sans aucun doute à expliquer pourquoi les systèmes d'exploitation temps-réel sont pour la plupart dédiés aux systèmes monoprocesseurs ! Le noyau Linux gère maintenant de manière efficace les systèmes SMP, mais n'a pas été conçu pour le support du temps-réel. Techniquement, seules des tâches temps-réel mou sont supportées via les ordonnancements FIFO et tourniquet (*round-robin*).
La solution ARTiS combine les avantages des systèmes généralistes et des systèmes temps-réel en confi-

gurant de manière asymétrique une plate-forme SMP Linux. L'objectif est de garantir à tous les processus l'accès aux services fournis par Linux, de conserver la possible utilisation de l'ensemble des ressources du SMP, tout en améliorant significativement le comportement temps-réel. La spécificité d'ARTiS est de partitionner les processeurs entre des processeurs temps-réel et des processeurs non temps-réel mais aussi de migrer les tâches qui tenteraient de désactiver la préemption sur un processeur temps-réel. ARTiS propose donc :
– un partitionnement des processeurs en deux ensembles ;
– différentes catégories de processus temps-réel ;
– un mécanisme de migration spécifique ;
– une politique d'équilibrage de la charge efficace ;
– des mécanismes de communications asymétrique.

**Partitionnement des processeurs en deux ensembles**

Un ensemble, dit NRT — non temps-réel, et un ensemble dit RT — temps-réel. Une politique d'ordonnancement spécifique est mise en œuvre sur chacun de ces ensembles. L'objectif étant d'assurer la meilleure latence d'interruption pour des processus désignés s'exécutant au sein de l'ensemble RT des CPU (processeurs).

**Deux classes de tâches temps-réel**

Ce sont des tâches temps-réel Linux traditionnels, ils diffèrent dans leur association aux processeurs :
– une, ou plusieurs[1], tâches Linux temps-réel sont attachées à chacun des CPU RT. Elles sont nommées **RT0**. Ce sont les tâches temps-réel de plus hautes priorités. Ces tâches ont la garantie que le processeur leur sera entièrement dédié tant que nécessaire. Ces tâches sont les seules autorisées à être non-préemptibles sur leur processeur. Cette propriété leur assure une latence minimale. Ces tâches RT0 sont les tâches temps-réel dur d'ARTiS. L'exécution de plus d'une tâche RT0 sur un même processeur est possible ; c'est alors au développeur d'assurer la faisabilité de l'ordonnancement de ces tâches pour que les latences puissent être garanties ;
– sur chacun des processeurs, d'autres tâches Linux temps-réel peuvent s'exécuter, mais **uniquement** en mode préemptible. En fonction de leur priorité, ces tâches sont nommées RT1, RT2... ou RT99. De manière générale, ces tâches sont désignées par tâches **RT1+**, tâches temps-réel de priorité 1 et plus. Elles peuvent profiter efficacement des ressources si les tâches RT0 ne consomment pas tout le temps CPU. Pour assurer la meilleure latence aux tâches RT0, les tâches RT1+ sont automatiquement migrées par l'ordonnanceur ARTiS sur un CPU NRT si elles sont sur le point de devenir non-préemptibles (quand elles appellent une des fonctions `preempt_disable()` ou `local_irq_disable()`). Les tâches RT1+ sont les tâches temps-réel mou d'ARTiS. Elles n'ont pas de garanties temporelles fermes, mais leurs demandes sont prises en compte selon une politique du meilleur effort (*best effort*). Elles sont aussi le support des parties les plus calculatoires des applications visées ;
– les autres tâches, non temps-réel, sont nommées « tâches Linux » dans la terminologie ARTiS. Elles ne sont liées à aucune contrainte temporelle. Elles peuvent coexister avec des tâches temps-réel et sont éligibles par l'ordonnanceur tant qu'aucune tâche temps-réel ne requiert le processeur. Comme pour les tâches RT1+, les tâches Linux sont automatiquement migrées vers un CPU NRT si elles tentent d'entrer dans une section de code non-préemptible ;
– les CPU NRT exécutent principalement des tâches Linux. Ils exécutent aussi les tâches RT1+ quand celles-ci sont non-préemptibles. Pour assurer l'équilibrage de charge du système, chacune de ces tâches peut être déplacée sur un processeur RT, mais seulement dans l'état préemptible. Quand une tâche RT1+ s'exécute sur un CPU NRT, elle conserve son niveau de priorité élevé par rapport aux tâches Linux.

**Mécanisme de migration spécifique**

L'objectif de ce mécanisme est d'assurer une latence aussi faible que possible pour les tâches RT0. Toute tâche RT1+ ou Linux s'exécutant sur un processeur RT est automatiquement migrée vers un processeur NRT quand elle tente de désactiver la préemption. Un changement fondamental par rapport au mé-

[1] ce qui a évolué depuis les premières définitions d'ARTiS

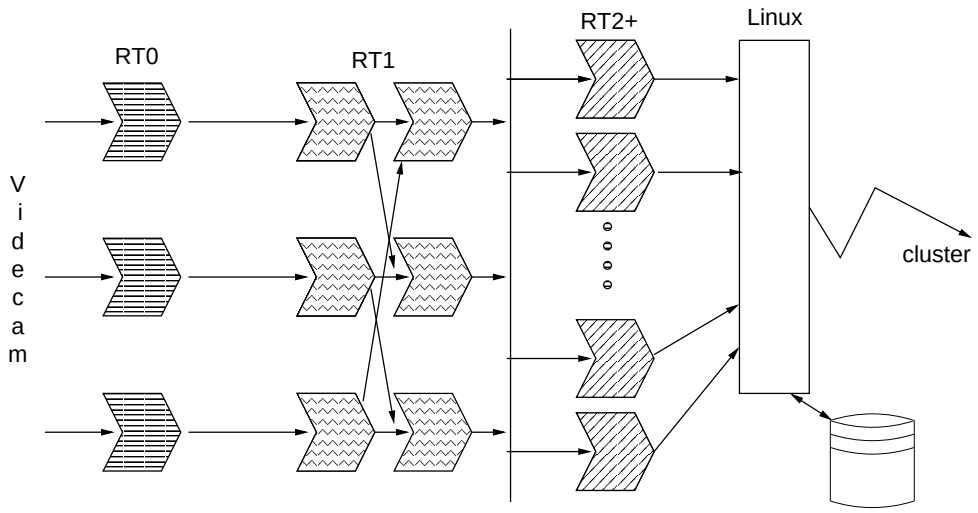


FIG. 1 – Une architecture de tâches d'une application ARTiS.

canisme original d'équilibrage de la charge de Linux est la nécessité d'éviter la prise de verrous inter-processeurs. Pour effectivement migrer les tâches, les CPU RT et NRT doivent communiquer via des queues. Nous avons implanté un mécanisme de queues FIFO sans verrou pour un lecteur/un écrivain assurant l'absence d'attentes actives de l'ordonnanceur ARTiS.

**Politique d'équilibrage de la charge efficace**

L'équilibrage de la charge assure la pleine exploitation d'un système SMP. Habituellement, l'équilibrage de la charge consiste à déplacer des tâches d'un processeur à un autre afin qu'aucun des processeurs ne soit oisif alors que des tâches sont activables sur un autre processeur. Dans le cas d'ARTiS, les spécificités du système doivent être prises en compte. Les tâches RT0 ne doivent jamais migrer, par définition. Les tâches RT1+ devraient revenir sur les CPU RT plus rapidement/prioritairement que les tâches Linux, ces processeurs offrant des garanties temps-réel non offertes par les CPU NRT. En vue de minimiser la latence sur les CPU RT et de fournir les meilleures performances au niveau du système global, des algorithmes spécifiques et asymétriques d'équilibrage de la charge ont été proposés [16].

**Mécanismes de communications asymétriques**

Au sein des machines SMP, les tâches échangent des données par lecture/écriture dans la mémoire partagée. Pour assurer une cohérence à ces échanges, des sections critiques sont nécessaires. Ces sections critiques sont protégées d'accès concurrents simultanés par des mécanismes de verrouillage. Ce schéma de communication n'est pas adapté à notre cas particulier : un échange de données entre une tâche RT0 et une tâche RT1+ va entraîner une migration de la tâche RT1+ avant que celle-ci ne prenne le verrou afin de l'empêcher d'entrer dans un état non-préemptible sur un CPU RT. Un schéma de communication asymétrique devrait par exemple utiliser des FIFO sans verrous dans un contexte un écrivain/un lecteur.

ARTiS supporte trois niveaux différents d'exécution temps-réel : RT0, RT1+ et Linux. Les contraintes des tâches RT0 visent à minimiser la variation de latence qui pourrait être due à une exécution non-préemptible sur le même processeur, cependant ces tâches restent des tâches utilisateur. Les tâches de l'ensemble RT1+ sont des tâches temps-réel mou, mais elles sont à même de profiter globalement du système SMP, en particulier pour des applications de calcul intensif. Elles sont aussi capables de déclencher des communications asymétriques qui évitent des migrations intempestives dues à des prises de verrous. Un mécanisme d'équilibrage de la charge garantie une bonne répartition sur l'ensemble des processeurs. Finalement, des tâches Linux peuvent venir s'exécuter, sans intrusion, sur les processeurs temps-réel. Elles sont donc elles aussi capables d'utiliser toute la puissance de la machine.

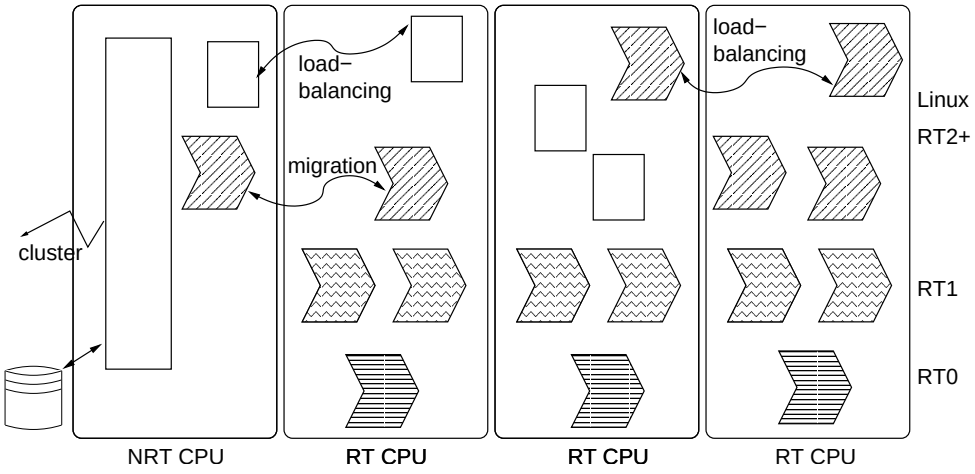


FIG. 2 – Placement des tâches sur les processeurs RT et NRT.

## 3. Déploiement d'applications sur ARTiS

Une application temps-réel ne prend tout son sens sur une machine SMP lorsqu'il y a à la fois des contraintes temps-réel et des besoins de calcul intensif. ARTiS est dédié à ce type d'applications. Les contraintes temps-réel sont assurées par les tâches RT0. Les communications entre ces tâches RT0 et les tâches RT1+ fournissent un flot de données vers la partie la plus calculatoire de l'application. Les tâches Linux assurent les traitements additionnels. L'équilibrage de la charge résulte en un agencement dynamique de ces différentes tâches sur les processeurs.

Le déploiement d'une application sur ARTiS nécessite l'identification d'un niveau particulier de temps-réel pour chacune des tâches et le placement de ces tâches sur chacun des processeurs de la machine. Ce déploiement peut être illustré par une application temps-réel de gestion de la qualité de production ; par exemple une application de détection des défauts sur une ligne de production continue s'exécutant sur une machine SMP de quatre processeurs, trois processeurs étant identifiés comme RT et le quatrième comme NRT. Plusieurs tâches sont présentes. La figure 1 illustre cette architecture de tâches :

- une caméra vidéo et/ou des capteurs reçoivent périodiquement des données. Jusqu'à trois tâches RT0 sont capables d'assurer l'acquisition des données avec des latences compatibles avec le temps-réel. Chacune de ces tâches est affectée à un CPU RT ;
- directement connecté à ces tâches, un calcul intensif sur des données régulières doit être mis en œuvre pour le traitement d'images. Un nombre fixe de tâches RT1 est dédié à cette activité data-parallèle, à la manière d'OpenMP. Chacune d'entre elles doit communiquer avec une tâche RT0 et d'autres tâches RT1 sans migrations inopportunes. Ces tâches sont elles aussi liées à un processeur, mais migreraient sur le processeur NRT si elles entraient dans une section de code non-préemptible. Elles tirent partie de l'architecture SMP de la machine ;
- les défauts sont ensuite identifiés à l'aide de structure de données irrégulières : le traitement de chacun des défauts est spécifique. Un nombre variable de tâches RT2 sont créées. Il leur est nécessaire de communiquer avec les tâches RT1 et entre elles. Parce que ce nombre de tâches est variable, la mise en œuvre de la politique d'équilibrage de la charge d'ARTiS est indispensable ;
- finalement, l'identification des défauts au regard d'une base de données, locale ou distante, est par exemple utilisé pour produire des statistiques... Cette étape ne requière pas de contraintes temporelles particulières et peut donc être menée à bien par des tâches Linux. Elles sont déployées sur le CPU NRT, mais peuvent utiliser les CPU RT quand ils sont inactifs.

La figure 2 illustre une possible répartition des tâches de l'application sur les CPU RT et NRT.

## 4. Évaluation de performances

Au cours de l'implantation d'ARTiS, nous avons effectué des tests de mesure afin d'évaluer les bénéfices apportés par l'approche en termes de latence d'interruption. Nous avons distingué deux types de

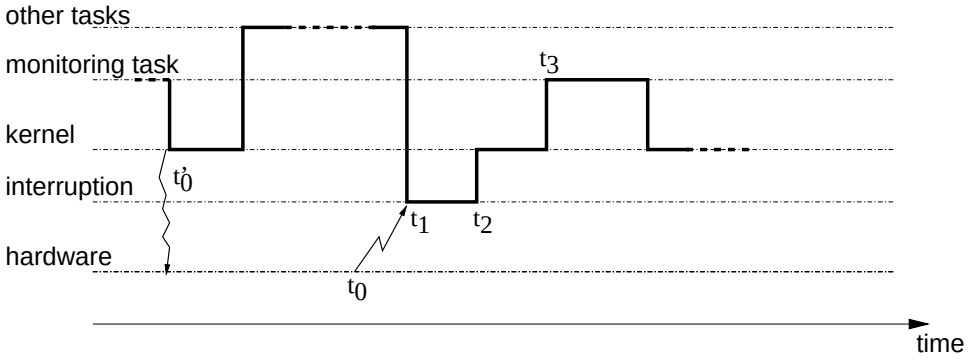


FIG. 3 – Chronogramme des différentes mesures associées à une interruption.

latences, une associée au noyau et une seconde associée aux tâches utilisateur.

**Méthode de mesure**

Les tests consistent à mesurer le temps écoulé entre la génération matérielle d'une interruption et l'exécution du code concernant cette interruption. Le protocole expérimental mis en place a été écrit en tentant de rester aussi près que possible des mécanismes habituels utilisés par les tâches temps-réel. La tâche de mesure règle un composant matériel de manière à ce qu'il génère une interruption à un temps connu précis, puis elle se met en attente de cette interruption. Lorsque l'interruption est reçue par le système, la tâche est réveillée et elle note le temps actuel. Puis le cycle recommence avec la mesure suivante. Cet enchaînement est particulièrement typique des applications temps-réel, qui attentent un évènement matériel, traitent les nouvelles données, envoient l'information et se remettent en attente. Il y a cinq instants différents associés à une interruption donnée, ils correspondent à différents emplacements dans du code exécuté (figure 3) :
– $t'_0$, programmation de l'interruption ;
– $t_0$, émission de l'interruption (instant précisé lors de sa programmation) ;
– $t_1$, entrée dans le gestionnaire d'interruptions spécifique à cette interruption ;
– $t_2$, fin du gestionnaire d'interruptions ;
– $t_3$, entrée dans la tâche utilisateur temps-réel.
Les mesures furent effectuées sur un quadri-processeurs Itanium II 1,3GHz. Le noyau Linux utilisé est une version 2.6.4 instrumentée. Le compteur sur lequel s'appuient toutes les mesures est l'`itc` (un registre du processeur qui compte les cycles) et les interruptions sont générées avec une précision au cycle près à l'aide du PMU (une unité interne à chaque processeur [15]).
Même lors de fortes charges du système, des états qui entraînent de longues latences sont extrêmement rarement exhibés. C'est pour cela qu'un grand nombre de mesures est nécessaire. Chaque test est composé de 300 millions de mesures, durant ainsi 8 heures. Lorsque les tests sont de cette durée les résultats sont tout à fait reproductibles d'une campagne de mesures à une autre.

**Type de latence d'interruption**

À partir des quatre points de mesure, deux valeurs intéressantes peuvent être calculées. Leur intérêt provient du fait qu'il est facilement possible de les associer à des techniques de programmation usuelles et aussi du fait de leur différences significatives selon les configurations testées. Ces deux types de latence sont :

**Latence noyau** $t_1 - t_0$. Il s'agit du temps écoulé entre la génération de l'interruption et l'entrée dans le gestionnaire d'interruptions (`pfm_interrupt_handler()` lors de ces mesures). Cette latence correspond à celle que subirait une tâche temps-réel écrite en tant que module noyau.

**Latence utilisateur** $t_3 - t_0$. Il s'agit du temps écoulé entre la génération de l'interruption et l'exécution du code associé dans la tâche utilisateur. Cela correspond à la latence que subirait une application temps-réel écrite entièrement en espace utilisateur. L'application reçoit les notifications à l'aide d'un appel système bloquant (un `read()`), ce qui est plus efficace que l'attente de signaux.

Les tâches temps-réel qui s'exécutent dans l'espace utilisateur ont été développées en utilisant l'interface traditionnelle et standard POSIX. Ceci est un des avantages principaux qu'ARTiS procure. Par conséquent, dans le contexte d'ARTiS, ce sont les latences utilisateur qui sont les plus importantes à étudier et à analyser.
Jusqu'à présent il n'a pas était fait mention de l'instant $t_2$. En fait, $t_2 - t_1$ représente la durée d'exécution du gestionnaire d'interruption. Cette durée s'est avérée particulièrement stable en fonction des configurations et elle est toujours très petite par rapport aux deux autres intervalles (environ $1\mu$s).

**Configuration ARTiS « idéal »**

Afin d'estimer les performances d'un système ARTiS tout en travaillant à sa réalisation, nous avons développé une configuration de simulation. Cette configuration de test est basée sur un noyau Linux vanilla et est asymétrique de la même manière qu'ARTiS. Dans la simulation un placement manuel et statique de chaque tâche sur le processeur sur-lequel ARTiS l'exécuterait est effectué :
– la tâche de mesure, équivalente à un processus RT0, est exécutée sur un processeur spécifique (un CPU RT) ;
– les tâches utilisant le processeur uniquement dans l'espace utilisateur, équivalentes à des processus RT1+ ou Linux qui ne désactivent jamais la préemption, sont exécutées sur chacun des processeurs ;
– toutes les autres tâches, équivalentes à des processus RT1+ ou Linux qui déactivent la préemption, sont exécutées sur un autre processeur spécifique (CPU NRT).
De plus, les interruptions qui ne sont pas utilisées par la tâche de mesure sont redirigées vers un CPU NRT. La répartition des tâches et des interruptions est prédéfinie et non-modifiable au cours des mesures.
Cet environnement purement statique est équivalent à la configuration d'un système ARTiS au moment précis où ces tâches spécifiques sont exécutées. À l'exception d'une légère modification du noyau évitant l'exécution de processus noyau sur le CPU RT, toute la configuration est définie depuis l'espace utilisateur. Du point de vue des mesures, cette configuration peut être considérée comme un ARTiS « idéal », aucune latence supplémentaire n'étant introduite par les migrations ou par l'équilibrage de charge.

**La configuration ARTiS actuelle**

Bien que l'implantation d'ARTiS ne soit pas encore achevée, la version actuelle implante la détection de fonctions menaçant le temps-réel : elle est capable de migrer automatiquement les tâches d'un CPU RT vers un CPU NRT. L'équilibrage de charge spécial n'est pas encore présent. Cependant le mécanisme original a été modifié de telle sorte qu'il n'est pas jamais effectué par un CPU RT, évitant ainsi les verrous inter-processeur. La migration d'un CPU NRT vers un CPU RT est encore présente afin de pouvoir utiliser les CPU RT s'ils deviennent oisifs. Vis-à-vis des mesures de latence, cette configuration devrait proposer des résultats très similaires à ceux de la version finale d'ARTiS.

**Environnement des mesures**

Les mesures ont été produites dans différentes configurations. Nous avons identifié trois paramètres indépendants qui affectent les latences d'interruption :

**Charge de la machine** Une machine peut être non-chargée (sans aucune charge) ou bien très chargée (tous les programmes décrits ci-après sont exécutés en même temps) ;

**Préemption noyau** Si la préemption noyau est activée, cette nouvelle option des noyaux Linux 2.6 permet de réordonnancer une tâche même lorsque du code noyau est en cours d'exécution. Cette configuration correspond à ce qui est couramment appelé le « noyau Linux préemptible » ;

**Partitionnement** Il peut être symétrique, la configuration standard, ou asymétrique, tel que dans la configuration ARTiS « idéal ».

Les huit configurations pouvant être obtenues à partir des combinaisons de ces trois paramètres ont été testées ainsi que la version actuelle d'ARTiS sur une machine chargée. À partir de ces résultats, nous avons sélectionné cinq combinaisons pour leur intérêt. Elles peuvent être construites de manière incrémentale en activant au fur et à mesure les trois paramètres : un noyau vanilla sans et avec charge, un noyau préemptif avec charge, une configuration ARTiS « idéal » chargée et l'implantation actuelle

d'ARTiS chargée. L'ensemble initial a été réduit parce que les configurations non-chargées ne présentent aucune latence élevée et qu'une configuration asymétrique sans préemption noyau ne représente rien de réaliste.

Lors des mesures, la charge système consiste à occuper le système à l'aide d'une utilisation intensive des processeurs et du déclenchement d'un certain nombre d'interruptions dans le but de maximiser la probabilité de verrous inter-processeurs et de désactivations de préemption. Pour cela, quatre programmes correspondant à quatre charges différentes sont utilisés :

**Charge de calcul** Une tâche exécutant une boucle infinie sans jamais faire d'appel système est attachée à chacun des processeurs, simulant ainsi une tâche de calcul.

**Charge d'entrée/sortie** Le programme `iodisk` lit et écrit continuellement sur le disque.

**Charge réseau** Le programme `ionet` inonde la carte réseau de requêtes à l'aide d'échos/réponses `ICMP`.

**Charge verrous** Le programme `ioctl` appelle la fonction `ioctl()` qui contient des accès au verrou noyau global (*big kernel lock*).

**Latences observées**

Les tableaux 1 et 2 résument les mesures réalisées pour chacune des configurations. Trois mesures sont associées à chacun des types de latence (noyau et utilisateur). La colonne « Maximum » correspond à la latence la plus longue observée au cours des 8 heures. Les deux autres colonnes contiennent la latence maximale des 99,999% (respectivement 99,999999%) plus petites mesures. Dans ces tests, cela équivaut à ne pas tenir compte des 3000 (resp. 3) plus mauvaises latences.

TAB. 1 – Latences noyau pour les différentes configurations.

| | | **Noyau** | | |
|---|---|---|---|---|
| **Configurations** | | 99.999% | 99.999999% | Maximum |
| Linux standard | non-chargé | 78$\mu$s | 94$\mu$s | 94$\mu$s |
| Linux standard | chargé | 77$\mu$s | 98$\mu$s | 101$\mu$s |
| Linux préemptif | chargé | 76$\mu$s | 98$\mu$s | 101$\mu$s |
| ARTiS « idéal » | chargé | 3$\mu$s | 8$\mu$s | 9$\mu$s |
| ARTiS | chargé | 71$\mu$s | 99$\mu$s | 101$\mu$s |

TAB. 2 – Latences utilisateur pour les différentes configurations.

| | | **Utilisateur** | | |
|---|---|---|---|---|
| **Configurations** | | 99.999% | 99.999999% | Maximum |
| Linux standard | non-chargé | 82$\mu$s | 174$\mu$s | 220$\mu$s |
| Linux standard | chargé | 2,8ms | 41ms | 42ms |
| Linux préemptif | chargé | 457$\mu$s | 29ms | 47ms |
| ARTiS « idéal » | chargé | 8$\mu$s | 27$\mu$s | 28$\mu$s |
| ARTiS | chargé | 91$\mu$s | 113$\mu$s | 120$\mu$s |

L'étude d'une configuration non-chargée n'apporte pas beaucoup d'information en elle même, mais sert de point de comparaison pour les autres configurations. En comparant cette configuration avec la même mais chargée on remarque d'abord que les latences noyaux ne sont pratiquement pas affectées par la charge. Par contre les latences utilisateur sont supérieures de plusieurs ordres de grandeur. Ceci est un problème typique de Linux, simplement parce qu'il n'a jamais été développé avec des contraintes temps-réel en tête.

La préemption noyau ne modifie pas les latences au niveau du noyau. On pouvait s'attendre à ce résultat puisque les modifications ne portent que sur un ordonnancement plus prompt des tâches, rien n'est changé pour améliorer les délais du coté noyau. En ce qui concerne les latences utilisateur, une réelle amélioration peut être notée : 99,999% des latences sont en-dessous de 457$\mu$s au lieu de 2,8ms. Malheureusement la valeur maximale des latences utilisateur est très similaire, de l'ordre de 40ms. Cette

amélioration permet du temps-réel mou avec de meilleurs résultats que le noyau standard mais en aucun cas elle ne permet le temps-réel dur, pour lequel même une unique latence au dessus du seuil fixé est inacceptable.
Dans la configuration « ARTiS idéal », les deux types de latences sont réduits de manière très notable. Il peut être même surprenant que les latences noyau soient inférieures à une configuration sans charge. Cela provient du fait que toutes les interruptions sont redirigées vers un processeur NRT et que par conséquent il est beaucoup moins fréquent que les interruptions soient masquées. Pour les latences utilisateur, l'amélioration est encore plus nette (le maximum passant de plus de 40ms à moins de 30$\mu$s).
Comme l'on pouvait s'y attendre, les résultats obtenus avec l'implantation d'ARTiS sont moins bons qu'avec la configuration « idéale » mais un maximum de 120$\mu$s pour les latences utilisateur reste encore bien en-dessous de la limite des 300$\mu$s qui était fixée. Ce système peut être considéré temps-réel dur, garantissant toujours de faibles latences aux applications temps-réel. La différence entre la version simulée et cette version d'ARTiS peut être expliquée par les temps nécessaires à la migration d'un CPU RT à un CPU NRT. Il se peut aussi que certaines latences soient dues à la migration de tâches d'un CPU NRT vers un CPU RT (ce qui ne devrait plus se produire dans l'implantation finale).
Ainsi, le noyau standard Linux, avec plus de 3000 temps de réponse supérieurs à 2ms, ne peut pas être considéré dans une perspective de temps-réel. Le gain obtenu avec un noyau préemptif peut permettre d'aborder le temps-réel souple. La version simulée d'ARTiS et son implantation actuelle assurent toutes deux des latences toujours inférieures à 300$\mu$s. Ce sont les deux seules configurations testées sur lesquelles un système temps-réel dur pourrait être basé.

## 5. Implantation actuelle d'ARTiS

Une API rudimentaire d'utilisation d'ARTiS a été définie. Elle autorise le déploiement d'applications sur l'implantation actuelle du modèle ARTiS définie comme une modification du noyau Linux 2.6. Une description plus détaillée de cette implantation est disponible dans [11].
L'utilisateur définit son application ARTiS en configurant le partitionnement des processeurs, et en identifiant les tâches temps-réel et leur affinité processeur via une interface `/proc` et quelques appels système (`sched_setscheduler()`...).
L'implantation actuelle consiste principalement en un mécanisme de migration assurant que seul du code préemptible est exécuté sur les processeurs RT. Ce mécanisme de migration repose sur des files de descripteur de tâches entre les processeurs. Cette structure, que nous nommons RT-Q, est implantée sans verrou partagé entre un processeur écrivain (un processeur RT) et un processeur lecteur (un processeur NRT) selon les algorithmes proposés par Vallois [18]
En dépit du fait que cette première implantation ne soit pas encore complète, en particulier au niveau des algorithmes d'équilibrage de la charge et des communications inter-processus, les premiers éléments de performance mettent en évidence des gains de latence significatifs. Ces résultats de l'implantation actuelle ne devraient pas évoluer, les mécanismes restant à mettre en œuvre ne fournissent que des fonctionnalités complémentaires et ne sont pas relatifs à l'amélioration de la latence. Les dernières mesures, faites sur une version de développement pratiquement finale, indiquent des latences très similaires à celles présentées ici. En conséquence, le système ARTiS peut être qualifié de système d'exploitation temps-réel dur.
Bien que l'ensemble des mécanismes d'ARTiS ne soit pas encore implanté, le système est d'ores et déjà utilisable. Il garantit effectivement à tout moment la possible préemption sur un processeur RT.
L'évolution principale de cette implantation concerne l'équilibrage de charge, mais nous étudions aussi le comportement des fils d'exécution du noyau (*kernel thread*) et leur influence sur les latences. Il apparaît évident que certains de ces fils d'exécution peuvent migrer ou même être liés à un CPU NRT, tel `kjournald` qui traite de la journalisation de systèmes de fichiers. D'autres doivent être remplacés, c'est par exemple le cas du fils d'exécution chargé, sur chaque processeur, de l'équilibrage de charge.

## 6. Conclusion

Nous avons identifié des applications pouvant tirer parti d'un système d'exploitation Open Source temps-réel capable d'exploiter la pleine puissance de machines multiprocesseurs sur lesquels des tâches

temps-réel cohabitent avec des tâches généralistes. Nous avons proposé ARTiS, un système basé sur un partitionnement du multiprocesseur en un ensemble de processeurs RT sur lesquels les tâches sont protégées des variations de latences, et un ensemble de processeurs NRT sur lesquels l'ensemble des activités pouvant entraîner des variations de latence sont exécutées. Ce partitionnement n'exclut pas un équilibrage des tâches sur la totalité de la machine, il implique uniquement une migration automatique des tâches quand elles sont sur le point de devenir non-préemptibles.

La première implantation d'ARTiS a été évaluée sur un quadri-processeurs Intel IA-64 et une latence maximale de 120$\mu$s a pu être garantie alors que le noyau Linux 2.6 standard produit des latences de l'ordre de 40ms, soit une amélioration d'un facteur 300.

ARTiS est implanté comme une modification du noyau Linux. Bien qu'incomplète, cette implantation est déjà utilisable. Les patches ARTiS pour le noyau Linux 2.6 pour architectures Intel x86 et IA-64 sont disponibles depuis la page web d'ARTiS [10].

Outre l'implantation complète, la validation de l'équilibrage de charge, et la mise en place de mécanismes de communications inter-processus asymétriques, les travaux futurs incluent aussi la définition d'ordonnanceurs locaux autres que l'actuel FIFO pour de potentiels multiples tâches RT0 sur un processeur.

**Bibliographie**

1. Brosky (Stephen). – *Symmetric Multiprocessing and Real-time in PowerMAX OS*. – White paper, Fort Lauderdale, FL, Concurrent Computer Corporation, 2002.
2. Brosky (Steve) et Rotolo (Steve). – Shielded processors: Guaranteeing sub-millisecond response in standard Linux. *In : Workshop on Parallel and Distributed Real-Time Systems, WPDRTS'03*. – Nice, France, avril 2003.
3. Bryant (Ray), Hartner (Bill), He (Qi) et Venkitachalam (Ganesh). – SMP scalability comparisons of Linux kernels 2.2.14 and 2.3.99. *In : 4th Annual Linux Showcase and Conference*. – Atlanta, GA, octobre 2000.
4. Chen (Paul). – *Implementing Basic Memory Protection in VxWorks: A Best Practices Guide*. – White paper, Alameda, CA, Wind River, 2003.
5. Cloutier (Pierre), Montegazza (Paolo), Papacharalambous (Steve), Soanes (Ian), Hughes (Stuart) et Yaghmour (Karim). – DIAPM-RTAI position paper. *In : Second Real Time Linux Workshop*. – Orlando, FL, novembre 2000.
6. Dougan (Cort) et Sherer (Matt). – *RTLinux POSIX API for IO on Real-time FIFOs and Shared Memory*. – White paper, Finite State Machine Labs, Inc., 2003.
7. Finite State Machine Labs, Inc. – RealTime Linux (RTLinux). `http://www.fsmlabs.com/`.
8. Gallmeister (Bill). – *POSIX.4, Programming for the Real World*. – O'Reilly & Associates, 1994.
9. Kuhn (Bernhard). – Interrupt priorisation in the Linux kernel, janvier 2004. `http://home.t-online.de/home/Bernhard_Kuhn/rtirq/20040304/rtirq.html`.
10. Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille. – ARTiS home page. `http://www.lifl.fr/west/artis/`.
11. Marquet (Philippe), Piel (Éric), Soula (Julien) et Dekeyser (Jean-Luc). – Implementation of ARTiS, an asymmetric real-time extension of SMP Linux. *In : Sixth Realtime Linux Workshop*. – Singapore, novembre 2004.
12. Momtchev (Momtchil) et Marquet (Philippe). – An asymmetric real-time scheduling for Linux. *In : Tenth International Workshop on Parallel and Distributed Real-Time Systems*. – Fort Lauderdale, FL, avril 2002.
13. Morgan (Kevin). – *Linux for Real-Time Systems: Strategies and Solutions*. – White paper, MontaVista Software, Inc., 2001.
14. Morgan (Kevin). – *Preemptible Linux: A Reality Check*. – White paper, MontaVista Software, Inc., 2001.
15. Mosberger (David) et Eranian (Stéphane). – *IA-64 Linux Kernel : Design and Implementation*. – Prentice-Hall, 2002.
16. Piel (Éric), Marquet (Philippe), Soula (Julien) et Dekeyser (Jean-Luc). – Load-balancing for a real-time system based on asymmetric multi-processing. *In : 16th Euromicro Conference on Real-Time Systems, WIP session*. – Catania, Italy, juin 2004.

17. Sillicon Graphics, Inc. – *REACT: Real-Time in IRIX*. – Technical report, Mountain View, CA, Sillicon Graphics, Inc., 1997.
18. Valois (John D.). – Implementing lock-free queues. *In : Proceedings of the Seventh International Conference on Parallel and Distributed Computing Systems*. – Las Vegas, NV, octobre 1994.
19. Williams (Clark). – Linux scheduler latency. *LinuxDevices.com*, mars 2002. – `http://www.linuxdevices.com/articles/AT8906594941.html`.
20. Yodaiken (Victor). – The RTLinux manifesto. *In : Proc. of the 5th Linux Expo*. – Raleigh, NC, mars 1999.
21. Yodaiken (Victor). – RTLinux beyond version 3. *In : Third Real-Time Linux Workshop*. – Milano, Italy, novembre 2001.