IPA

2022 年度 未踏 IT 人材発掘・育成事業 採択案件評価書

1. 担当 PM

田中 邦裕（さくらインターネット株式会社　代表取締役社長）

2. クリエータ氏名

飯田 圭祐（慶應義塾大学 環境情報学部環境情報学科）
柚山 大哉（慶應義塾大学 環境情報学部環境情報学科）

3. 委託金支払額

2,736,000 円

4. テーマ名

複数の ARM マシンを一つに集約するハードウェア仮想化レイヤ

5. 関連 Web サイト

ソースコード：https://github.com/k-mrm/Pilevisor

6. テーマ概要

ビッグデータ時代の到来により、生物学におけるゲノム解析などをはじめとする様々な分野で大規模なデータを扱う機会が増加している。これらの影響で、計算機科学の専門家以外でもクラスタリング環境を使用する機会が増えている。英 Arm は新しいアーキテクチャである Armv9 と、Intel Xeon に対抗する省電力・高性能なプロセッサ Arm Neoverse を発表しており、これからサーバにおける ARM アーキテクチャの割合が増えてくると見られているため、ARM マシンで構成されたクラスタリング環境の需要が増えると予想される。

既存のクラスタリング手法においては、ミドルウェアを用いて複数の物理マシンに跨っている CPU、メモリを活用していた。この手法では、ミドルウェアのインタフェースに従ったプログラミングを習得する必要があり、計算機科学の専門家ではない人々にとってハードルが高い。

本プロジェクトでは複数の物理 ARM マシンに跨って動作するハイパーバイザを実装する。実装されるハイパーバイザ上で動作する VM は、ゲスト OS からは集約された一つの物理マシンとして認識され、その上で動作するアプリケーションはクラスタリング環境であることを意識することなくリソースにアクセ

スできるようにする。つまり、クラスタリング環境上で既存の OS・アプリケーションがそのまま動作するようにする。具体的には、クラスタのノードとして Raspberry Pi 4、ゲスト OS として Linux を想定した上で、以下の実現を目標とする。

1. 複数のノード上に実装したハイパーバイザ上でゲスト OS が動く。
2. ゲスト OS が集約された CPU と RAM を認識して透過的にアクセスができる。
3. ゲスト OS 上で動くマルチスレッドアプリケーションが本システムで動く。
4. 本システムでスケールするアプリケーションとスケールしないアプリケーションを示す。
5. 実行中にノードが追加されたときにゲスト OS が増えた CPU と RAM を認識できる。

本プロジェクトの成果により、計算機科学の専門家以外でも容易にクラスタリングによる処理性能の向上ができるだけでなく、あらゆる分野の研究・開発の高効率化に寄与することができると考える。

7. 採択理由

本プロジェクトでは、昨今の性能向上と普及が著しい ARM マシンを複数組み合わせて、手軽に 1 つのコンピュータクラスタとしてユーザが利用できるプラットフォームを構築する。
スケールアウトできるコンピュータクラスタは人類のひとつの夢とも言え、過去から何度もチャレンジがなされてきたが、ARM という比較的最近に普及してきたデバイスを、モダンな手法でクラスタにするというのは目新しい。

ただ、既存のアプリケーションに手を入れずにクラスタ上で稼働させる事は非常に困難であり、アーキテクチャへの理解度やプログラミング能力といった技術力だけでなく、どの CPU とメモリ空間にユーザプログラムを配置するかや、筐体を超えたメモリアクセスの保障、動的なスケールアウト・スケールインの手法など、アイデアとアルゴリズムの設計力も重要になってくる。

このように、完成させられれば影響力が大きいが、困難性は高い、という未踏のプロジェクトである。その上で提案者らの実績から、難しいが達成は可能という判断をもって採択することとした。

8. 開発目標

本プロジェクトは、複数の物理 ARM マシンと Gigabit Ethernet で構成されるクラスタに跨って動作するハイパーバイザを実装し、クラスタ上の資源を集約した単一のシステムイメージを提供することを目標とした。これにより、OS や

アプリケーションを高速化することができ、資源の単一化により、プロセスなどの資源管理が簡便化する。ゲスト OS の種類を限定しないため、一般的な OS・アプリケーションを改変することなくそのまま動作できる。

比較対象のプロジェクトとして、MOSIX や Kerrighed などが挙げられる。これらのプロジェクトは、Linux を分散 OS に改造してクラスタ上の資源を集約した単一のシステムイメージを提供していた。しかし、Linux しか動作しない、Linux の書き換えが必要、Linux のアップデートに追従する必要があるという問題があった。また、vNUMA や Virtual Multiprocessor、GiantVM、vSMP などは、それぞれ特定のハードウェアに対応した実装となっており、改変が必要な場合があったり、メモリの集約が行われなかったり、ホスト OS のオーバーヘッドがかかるなどの欠点があった。

これらに対して、本プロジェクトでは、ARM アーキテクチャ向けに実装されたハイパーバイザ「Pilevisor」を提供することで、既存のエコシステムを利用しながらクラスタ上の資源を効率的に活用することを目指した。

9. 進捗概要

本プロジェクトでは、ARM ハードウェア仮想化技術を使用して、複数の ARM マシンを 1 つの仮想マシンに集約するためのハイパーバイザを実装した。これにより、クラスタ全体の資源を仮想マシンで利用できるようになり、既存のアプリケーションをクラスタ上で再利用できるようになった。

ハイパーバイザは、C 言語と ARM アセンブリ言語を使用してフルスクラッチで実装された。また、デバイスドライバ、ノード間通信プロトコルスタック、CPU 集約機構、仮想電源コントローラ、仮想割り込みコントローラ、メモリ集約機構、IO フォワーディング、ノード間のクロック同期など、様々な機能が実装された。

ARMv8 には、CPU の特権レベルとして Exception Level（EL）が存在し、0〜3 の 4 レベルに分かれている。EL0 から数字が上がるにつれて権限が強くなって、アクセス可能なレジスタがより増えていき、各 EL で実行するソフトウェアの種類は、表 1 の通りである。

表 1：各 EL で動作するソフトウェアの種類

EL	ソフトウェア種別
EL0	ユーザモードのソフトウェア
EL1	カーネルモードのソフトウェア
EL2	ハイパーバイザ
EL3	トラストゾーン

本プロジェクトでは、ARMv8 アーキテクチャ向けに EL2 上に構築されたハイ

パーバイザの実装を行った。このハイパーバイザは、ゲスト OS が必要な場合にのみトラップして動作するため、演算命令など、ハイパーバイザの介入が必要のない場合は何も処理を行わずに済む。ハイパーバイザの介入が必要な場合には、ARMv8 の仮想化支援機構を利用して高速にトラップを行い、ハイパーバイザの処理を行うことができる。

EL2 は EL1 よりも強い権限を持っており、EL2 で実行されるハイパーバイザは、EL1 で稼働するホスト OS の保護を行うことができる。また、EL2 は EL1 の割り込みをトラップすることができ、EL2 からホスト OS に割り込みを転送することができる。この機能を利用することで、ハイパーバイザはゲスト OS が発生させた割り込みをトラップし、必要に応じてホスト OS に割り込みを転送することで、ゲスト OS とホスト OS の間で割り込みを共有することができる。

ハイパーバイザは薄いハイパーバイザであり、単一の仮想マシンのみをサポートする。この薄いハイパーバイザの実装により、仮想化によるオーバーヘッドを最小限に抑えることができる。

メモリアクセスについては、自ノードへのメモリアクセスと他ノードへのメモリアクセスを区別するために、2 段階ページ変換を使用している。これは、ゲスト空間から見た物理アドレスと実際の物理アドレスの変換を透過的に行う仮想化支援機構の一つである。ARM では、Stage 2 Page Translation と呼ばれる。2 段階変換ページテーブルを使用して、仮想共有メモリの実装を行っている。このページテーブルは、ページサイズ（4096 バイト）単位でマッピングを行い、他ノードから受け取ったメモリページをキャッシュする。また、メモリの owner を追跡し、キャッシュ一貫性制御を保つ仮想共有メモリマネージャを実装している。

次にノード間通信であるが、Ethernet を介してノード間でメッセージを伝える必要があり、そのためにノード間通信プロトコルを制定し、プロトコルスタックの実装を行った（表 2）。送信メッセージのサイズが最大 4160 バイトとなるため、Ethernet フレームのデフォルト上限サイズである 1536 バイトを超えるため、ジャンボフレームを使用することにした。メッセージは 17 種類あり、それぞれ異なる目的を持っている。ペイロードに Ethernet ヘッダを付ける方法については、Pilevisor ではポインタ演算と最小限のコピーのみでヘッダの追加・削除ができるデータ構造を実装しており、処理のオーバーヘッドを最小限に抑えている。

表 2：メッセージの種類

メッセージ名	概要
init	発見ブロードキャスト
init ack	init への返信
cluster info	クラスタ情報のブロードキャスト

setup done	サブノードの初期化完了通知
boot sig	仮想 CPU0 起動通知
cpu wakeup	CPU 起床命令
cpu wakeup ack	cpu wakeup への返信
fetch	ページフェッチ命令
fetch reply	fetch への返信
invalidate	ページへの invalidate 命令
invalidate ack	invalidate への返信
interrupt	他ノード CPU への割り込み通知
gic config	GIC の状態通知
mmio request	他ノード CPU への MMIO アクセス
mmio reply	mmio request への返信
sgi	CPU 間割り込みの通知
panic	システム異常通知

各ノードが起動したときは、自分以外のノードが誰なのか、自分はどこのノード担当なのか分かる術を持っていない。そのため、最初に認識のための通信を行って、各ノードがクラスタの状態を知る必要がある。ノード認識通信では、6 つのステップで行われる。最初に、メインノードが LAN 内にいる全員に発見ブロードキャストを行い、その後、サブノードが応答して各種情報を送信する。その情報を元に、クラスタ情報を構築し、全ノードにブロードキャストする。その後、各ノードはクラスタ情報を元に初期化を行い、初期化終了通知をメインノードに送信する。メインノードは、全ノードから setup done が返ってくるまで待ち、前準備が行ったら、boot sig メッセージをブロードキャストする。最後に、sgi を受け取ったノードは、割り込みコントローラの機能を使用して、ゲスト OS に仮想割り込みを注入する。また、物理 CPU の CPU 番号はノードごとに 0 から割り当てられているため、仮想マシンが直接物理 CPU の CPU 番号を直接参照すると、矛盾が生じる。そのため、ノード間認識通信の時に決定された仮想 CPU 番号を仮想マシンに見せるようにする必要があり、ARMv8 の仮想化支援機構を使用した。

これらの実装により、複数のノードに跨った ARM の実機上で、単一の仮想マシンを実現することを達成した。

10. プロジェクト評価

本プロジェクトは、ARM マシンのクラスタの資源を仮想化技術によって集約し、単一の仮想マシン上で複数のオペレーティングシステムが同時に動作することを可能にすること目指しており、複数のマシンを組み合わせて、より効率的

にリソースを利用することができるようになった。

このプロジェクトの最も大きな成果は、ARM アーキテクチャに対してハイパーバイザを適用し、単一システムイメージを提供する仕組みを実現したことであり、Linux のアプリケーションを改変なく動作させることに成功し、非常に実用的な価値があることが示された。

なおクリエータ自身は、これまで「車輪の再発明しかしてこなかった」と述べており、実際に既存の実装を改良することでさまざまな開発を行なってきたようであるが、今回のプロジェクトにおいては先行の事例が極めて少ない中で、試行錯誤をしながらさまざまな新しい実装を進めてきたことが素晴らしいと言える。

実際にプロジェクトの途中では実装が進まない時期もあり、かつ情緒面でも心配されることがあったり、実装が危ぶまれたりしたこともあったが、典型的な追い込み型で成果報告会の発表直前に実機稼働を成し遂げるという、ヒヤヒヤしながらも結果の出せるプロジェクトになったと言えよう。

これらの成果物はオープンソースで公開しており、他の研究者や開発者がこれを利用して、より高度な仮想化技術を実現することができるようになる。これにより、ARM マシンを用いた様々なアプリケーションやシステムの開発が促進され、より効率的かつスケーラブルなシステムの構築が可能になることが期待される。

11. 今後の課題

本プロジェクトの今後の課題としては、システムの改良、スケーラビリティやパフォーマンスの向上、ユーザビリティの向上などがあげられる。

まずシステムの改良としては、仮想共有メモリのバグ修正が必要であり、現在 Raspberry Pi 4B で本システムを動かした際に、仮想共有メモリがハングすることがあるが、すでに原因は特定済みであるため、それをもとに修正を行う必要がある。また、仮想共有メモリのアルゴリズム改善も必要である。現在の実装では仮想共有メモリはメモリの書き込みが 1 つのノードしかできないため、書き込み操作がボトルネックになっているが、すでに分散共有メモリアルゴリズムとして複数ノードが同時に書き込み可能なアルゴリズムがあるため、それを実装することでパフォーマンスの改善やスケーラビリティの向上を目指す。

次にスケーラビリティやパフォーマンスの改善であるが、前出の共有メモリの書き込みアルゴリズムの改良のほか、ネットワークのボトルネック改善や、コードのリファクタリングなどを進め、ノードを増やせば増やすだけスケールアウトできる性能の実現が望まれる。

最後にユーザビリティの向上であるが、Pilevisor はまだ開発途中のプロジェクトであり、現在は専門知識を持った技術者向けの利用が主ではあるが、一般ユーザが簡単に利用できるような UI やドキュメンテーションを整備することで、

より広い層での利用が可能になることが期待される。