動的なノード構成変化に適応した負荷分散可能なOSの提案

赤木 永治[†]　中村 豊[‡]　藤川 和利[‡]　砂原 秀樹[‡]

概要: 近年、多数の計算機が企業や研究機関、一般家庭に導入され、Grid コンピューティングなどによる計算能力の有効活用が注目されている。しかしながらそれらの分散処理技術は、計算機のネットワーク接続状況や稼働状態が頻繁に変化するような場面では有効に機能しない。また利用には専門的知識・技術が必要であることや、その環境専用のプログラムが必要となるなど、一般的な利用者は容易に使うことができない。さらに汎用オペレーティングシステムの上で動作するため、そのオーバーヘッドによる性能低下が避けられない。そこで、本稿では CPU 資源を共有し、計算機群の状態変化に柔軟に適応して最大限の資源を活用できること、利用者が意識せずとも自動的に分散処理が行え、しかも処理性能の高い分散処理環境を提供することを目的とした分散処理専用の OS を提案する。

キーワード: オペレーティングシステム、Grid コンピューティング、分散処理

A Compact Distributed Operating System for Dynamically Configurable Cluster Environment

AKAGI, Eiji[†]　NAKAMURA, Yutaka[‡]　FUJIKAWA, Kazutoshi[‡]　SUNAHARA, Hideki[‡]

Abstract: There is much interest in Grid Computing due to its ability to make effective use of the CPU processing power amongst several computers. However, systems are not flexible enough to handle the addition or removal of computer clients and it is also unable to adjust itself according to the dynamic network conditions, and they generally require knowledge and skills of its efficient use. For general users, it is too difficult that they develop customized programs only for the system. Furthermore, the overhead causes performance degradation for the systems based on market available OSs. This paper discuss the recollection of the maximum possible processing resources from a pool of computers under dynamic conditions. We propose a compact and self-organized OS which is able to provide the users in a distributed environment with the best possible processing power available with no special configuration and the ease of management.

Keywords: Operating System, Grid Computing, Distributed Computing

1　はじめに

計算機の小型軽量化および価格低下が進み、企業や研究機関、さらには家庭にまで多数の計算機が導入され、複数の計算機を一度に利用できるようになった。また計算機の必要とされる応用分野はますます拡大しており、多種多様な場所や目的で利用されている。その中にはゲノム解析や高エネルギー物理学、実時間三次元レンダリングなど大量の計算能力を必要とする分野も多く、更なる計算能力の向上が求められている。

それに対し計算機単体の能力向上には限界があるため、多数の計算機を連携させ負荷分散することで性能向上を図る試みが Grid コンピューティングや分散オペレーティングシステムなどの分散処理技術である。これらの技術は、従来その専門家が導入・運用していた。しかし応用分野が情報技術以外の研究機関や企業、個人ユースなどへ広がるにつれ、分散処理の専門家ではない一般的な利用者にも使用されるようになった。

一方で、専門家ではない一般的な利用者が分散処理環境を利用するには、現状いくつかの問題がある。例えば、Grid など分散処理環境の設定や運用には専門的な技術が必要であり、容易に利用できるも

[†]奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science,
Nara Institute of Science and Technology
[‡]奈良先端科学技術大学院大学 情報科学センター
Information Technology Center,
Nara Institute of Science and Technology

のではない。また、その環境に合わせたソフトウェアの開発や導入を必要とし、普段利用している計算機用のプログラムは動作しない。さらに一般向けには、いつでもどこでも簡単に利用できること、分散環境を意識しなくても利用できること、軽快に利用できることが重要であるが実現できていない。

一般利用者が分散処理環境を容易に利用できるためには、

- 設定や運用が簡便で、いつどこでも利用できる
- 普段使用している環境をそのまま利用できる
- 即座に利用でき、コンパクトで高速に動作する

という特徴を持つ分散処理環境が必要であり、例えばフロッピーディスク一枚で OS が起動して自動的に設定・構成されるのが理想である。実現方法としては、Grid システムの拡張、汎用 OS の機能追加、新しい OS の開発が考えられる。しかし、Grid システムは既存 OS 上に構築されるためコンパクトにできない。また、汎用 OS では性能を追求するために、OS の基本設計の方針変更が必要なため困難である。よって、分散処理環境の基盤となるコンパクトな OS を新たに開発するのが適切だと考えられる。そこで本論文では、余計な機能を排して分散処理環境向けに特化したオペレーティング・システム (OS) である SiON [1]を提案する。

以下、本論文の構成を述べる。2 節では分散処理の要件を挙げて既存技術を検討し、その問題点を指摘する。3 節では SiON の設計について述べ、4 節ではその実装を説明する。さらに 5 節ではプロトタイプの有効性検証を行う。最後に 6 節でまとめと今後の課題について述べる。

2 既存の分散処理技術と問題点

本節では、まず分散処理技術を一般利用者へ普及させるための要件を明らかにする。その後、分散処理に関わる既存技術として Grid コンピューティングと分散オペレーティングシステムを取り上げ、その特徴と問題点を考察する。

[1] Self Integrated Operating-system for Nodes

2.1 一般利用者の要件

一般利用者が分散処理技術を容易に利用するために求められる要件について議論する。なお、以降では分散処理環境内にある計算機をノードと呼ぶ。

- 運用性:
  組織内ネットワークやインターネットなど、ネットワークへの接続状況や計算機の稼動状態が常に変化するような環境でも、有効に機能する分散処理環境が必要である。また、分散処理に使用される多数の計算機の設定や管理が専門的な技術を持たない利用者にも容易に行なえることが求められる。
- 親和性:
  利用者が普段慣れ親しんでいる計算機環境およびプログラムをそのまま利用でき、管理サーバや処理を分散するノードを意識しなくてすむように、分散処理環境は既存環境との親和性を確保する必要がある。
- 即応性:
  利用者にとって、ネットワークに接続すればすぐに利用可能になる、操作に対して即座に応答が返ってくるなどの即応性が重要である。

以下、Grid コンピューティング (OS 上に分散処理機能を追加) と、分散 OS (OS 内に分散処理機能を有す) を考察する。

2.2 Grid コンピューティング

Grid コンピューティングとして著名なものには、Globus[1], Condor[2], UNICORE[3] などがある。また、SCore[4] や Beowulf[5] も同じ方式と言える。これらのシステムの特徴としては、汎用 OS 上で稼働し既存の計算機をそのまま流用して大規模な分散環境を構築可能なことである。これによりハードウェアの投資コストが抑えられるとともに、汎用 OS の多機能性や高度なサービスをそのまま利用可能である。また全資源を集中管理することで、高度な並列化や最適なオブジェクト・マイグレーションを可能としている。

しかしながら、汎用 OS 上に構築されるためにその OS の制限を受ける。例えば、汎用ゆえに様々な

機能を備えているため、分散処理には不要な機能がオーバヘッドを生じさせる。また、全計算機にソフトウェアを導入して設定・管理を行う必要があり、高機能になるほど逆にその繁雑さ・難しさが一般利用者にとって利用の妨げになっている。さらに今後の利用範囲拡大に伴い、計算機の接続・離脱や稼働状態が頻繁に変化するところでは、集中管理に重要な構成ノードの登録が困難になり、結果として最適な Grid を構成できない。

他方、Grid 上でプログラムを実行するためには、その環境に合わせた分散処理用の専用ライブラリをリンクしなければならない。また、プログラムの実行用コマンドが複雑で使いこなせないため結局は rsh で明示的に分散させているというように、使いにくさの問題がある。

2.3 分散 OS

比較的昔から研究されている技術に分散 OS が挙げられる。分散 OS は多種多様で特徴や機能も様々である。ここでは負荷分散について考察する。

著名なものに Amoeba[6][7]、Mach[6]、Plan9[8] などがある。また、SSS-PC[9] など、クラスタとしての利用を目的としたものも多い。

これらは、最初から分散処理を考慮しており、OS としてその機能を有している。そのため、分散処理に必要な設定や手間は少なく、同じ OS 間では高い親和性と自動化、透過性の確保が実現されている。

しかし、クラスタ向け以外は性能的に不十分である。また、その OS 専用のアプリケーションプログラムが必要であり、利用者が環境を移行しなければならない。利用者にとって、新しい OS やその環境を習得することは一般に大変な労力であるとともに、使い慣れたアプリケーションプログラムが利用できないといった不便さがある。これは、Windows や UNIX が普及している現在では難しい要求である。また、OS の導入や設定には専門知識や労力が必要であることも問題である。

2.4 問題点のまとめ

分散処理環境を提供する既存技術について、以下にその問題点をまとめる。

- 運用性が低い：Grid や分散 OS の導入および設定作業には、専門的な技術が必要である。また、運用管理には時間的、人的コストがかかる。さらに、ノードの接続状況や稼動状態の変化への適応が不十分である。
- 親和性がない：利用する分散処理環境に合わせたプログラムが必要であり、一般利用者には開発が難しい。
- 即応性が低い：既存の OS 上に分散処理能力を構築することにより、オーバーヘッドが生じて処理能力が低下する。また複雑で肥大化したシステムでは最適化も困難であり、容易に利用できない。

3 SiON の設計

3.1 設計目標

2 節で挙げた問題点を解決するため、本研究では「運用性の向上」、「親和性の実現」、「即応性の確保」を設計目標とする。

これらの目標のためには、スケジューラやプロセス間通信などの必要最小限の機能に絞りこむと同時に、個々の機能を簡潔にして性能を向上する必要がある。また、運用性を高めるためにはフロッピーディスク一枚で OS が起動でき、起動後に設定や管理作業をしなくても自動的に分散処理環境を構成できるようなシステムでなければならない。

Grid システムは既存 OS の上に構築されるため、どうしてもコンパクトにはならない。また、スケジューラやプロセス間通信は OS の基本設計に関わる根幹部分であり、既存の汎用 OS を改造することでは抜本的な性能向上は難しいと考えられる。そこで我々は、できるだけコンパクトで高速な、分散処理環境の構成ノード専用の新たな OS、SiON を提案する。

3.2 SiON の概要

3.2.1 利用形態およびシステム構成

SiON が形成する分散処理環境および利用形態を図 1 に示す。利用者は、まず自分の計算機をローカルネットワークに接続する。この計算機では既存 OS (例えば Linux) が稼動しており普通に利用でき

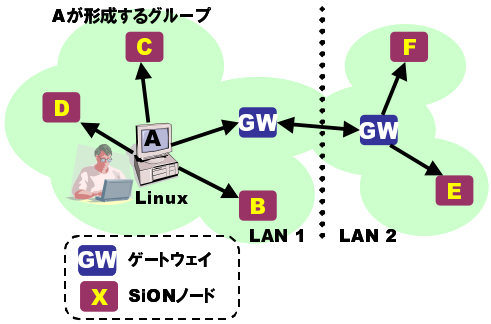

図 1: 利用形態

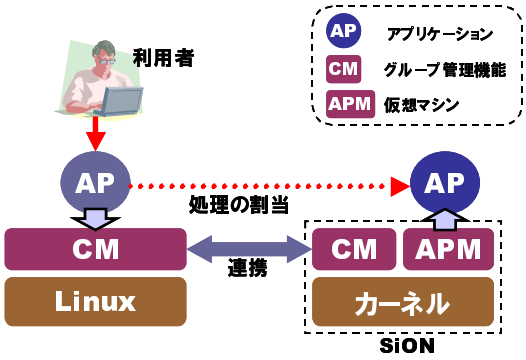

図 2: システム構成

る。サブネット内には SiON が稼動するいくつかのノードが存在し、互いに連携して分散処理環境としてのグループを形成している。利用者の計算機には SiON と連携するためのグループ管理機能をパッケージなどで導入しておき、起動しておく (図 2)。この機能は、ネットワークに接続した段階で SiON のノード群と連携して自動的にグループを構成する。特別な設定は必要ない。

図 1 のように、利用者が自分の計算機 A からアプリケーションを実行すると、A で稼動中のグループ管理機能がグループ内から実行に最適な計算機 B を選択し、処理を割り当てる。

また、図 1 の右方に示すようにサブネット外の環境と相互通信できるゲートウェイを用意することで、広範囲なノード群からなる分散処理環境も構築可能とする。

3.2.2 運用性：グループの形成と維持管理

SiON では資源を共有するグループを動的に管理することで環境変化に柔軟に適応できるようにする。このため、グループの形成や維持管理は SiON での重要な要素であり、グルーピングアルゴリズムが性能や機能に影響する。表 1 に 4 つのグループ管理方式の比較を示す。

表 1: グループ管理方式の比較

	集中管理	階層管理	ノード分散	P2P 分散
構成変化適応力	×	×	○	○
即応性	○	△	○	×
耐故障性	×	△	○	○
管理コスト	×	×	○	○
scalability	×	○	×	○
ノード負荷	サーバ高	サーバ高	高	中
全体最適化	○	△	△	×
ノード管理	集中管理	集中管理	ノード	ノード
適用規模	中	大	小	大

集中管理方式は、分散処理用の計算機群を集中的に管理するサーバを一台用意し、計算機全体のグループ形成や負荷分散を行なわせるものである。階層管理方式は、集中管理方式のグループを複数集めて階層的に構成するもので、より上位の管理サーバが下位のグループ群を管理するものである。ノード分散方式とは、各ノードが必要なグループを自分で形成し維持管理する方法である。P2P 分散方式は、各ノードが少数の他ノードの情報を持ち、それを順に辿ることで連鎖的に繋がる多数のノードとのグループ形成を可能とするものを表す。

評価項目は、本研究において重要な順に上から並べてある。これらの他にもモバイルノードに適したグルーピング手法など様々な方式があるが、SiON では各ノードに自律的なグループ形成をさせるためと実装が容易であるとの理由で、ローカルネットワーク内のグループ形成にノード分散方式を用いる。また、ネットワーク間のグループ間連携には、中継に特化して即応性を上げたゲートウェイに P2P 分散方式を適用し、スケーラビリティを確保する。

次に、グループ管理の実際を述べる。共有資源の中から最適なものを選択するため、SiON ではタスクを割り当てる側のノード (依頼元ノードと呼ぶ) が自分の責任で割り当てられる側のノード (請負ノードと呼ぶ) を選択するようにした。

具体的には、最初にグループリストを確認して実行したい処理に適したノードをいくつか選択し、処理割り当て要求を送付する。

それを受信したノードは、その内容を確認するとともに自分の負荷状態、実行環境、依頼内容に合致するかどうかを判断し問題なければ “許可” を、合

致しなければ “不許可” を自分の負荷状況と合わせて依頼元ノードへ返す。

次に、各請負ノードから返された情報を確認し、割り当て可能なノードの中から CPU 負荷やメモリ容量などを検討して最適な割当先ノードを選択する。その後選択されたノードへプログラムを送付して実行してもらう。

なお、このグループ管理機能は SiON に組み込まれるものであるが、同じ機能を利用者側の計算機 (既存 OS) でもデーモンとして動作させる。これにより、利用者の計算機を分散処理環境の一ノードとして組込むことができるようにする。

3.2.3 親和性：仮想マシン

既存 OS のプログラムをそのまま動作可能とするため、SiON では利用したい既存 OS をエミュレートする仮想マシンを提供する。

この仮想マシンは次のような特徴を持つ。

- 既存 OS のシステムコールをエミュレート
  既存 OS のシステムコールをハンドリングし、要求にあったサービスを仮想マシンがプログラムに提供する。
- アプリケーションはシングルタスクで動作
  仮想マシン上では、アプリケーションプログラムはシングルタスクで動作する。処理効率を極力高めるため、そのプログラムの実行に集中して完了するまで中断しない。
- 独自のスケジューラ
  カーネルとは別に、既存 OS 用プログラムの実行を制御できるように、独自のスケジューラを持つ。SiON の仮想マシンはシングルタスクであり、そのスケジューラはプログラムに実行権を引き継いでその完了を待つ。

なお、分散処理の目的の一つである並列処理に関して、SiON ではまだ検討が不十分である。この仮想マシンをどのように並列向けに構成するかは今後の課題である。

3.2.4 即応性：簡素化と高速化

構造を簡素化し、高速な動作が行えるように以下のような設計にした。

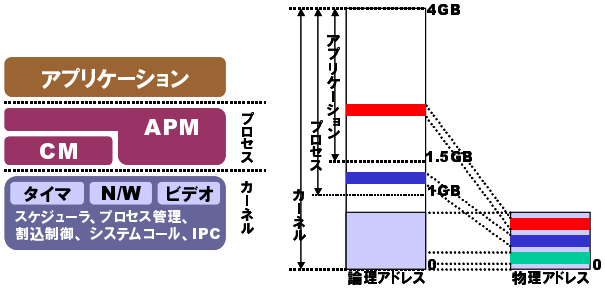

図 3: システム構成およびメモリマップ

- 最小限のモジュール構成
  SiON は、可能な限りコンパクトで高速なカーネルを基礎に、グループ形成を行うプロセスと仮想マシンを提供するプロセスを動作させるようにした。これにより、モジュール性を高めて機能拡張や改良を容易にした。同時にプロセス間通信機構を極力簡素化および高速化することで、モジュール化によるオーバーヘッドの低減を図った。
- カーネルサービスの絞り込み
  カーネルは、スケジューラ、プロセス管理、割込み制御、デバイス管理およびプロセス間通信機構のみ行なう。またデバイスとしてはタイマ、ネットワークおよびビデオのみサポートする。これによりシンプルで高速なカーネルを構成する。

4 SiON の実装

現在、SiON を IBM PC/AT 互換機上に実装中であり、ネットワーク周りを除くカーネルまで完成している。すなわち、4.2 節の「実装の詳細」のうち、「グループ形成と維持管理」および「仮想マシン」についてはまだ実装されていない。

4.1 システム構成とメモリマップ

最初にそのシステム構成を図 3 左に示す。構成はカーネル、プロセス、アプリケーションの大きく 3 つに分けられる。カーネルはさらにプロセス管理、スケジューラなどの基本部とタイマなどのデバイスドライバから成る。プロセスはカーネルで提供されない機能を担当する部分であり、コンパニオン・

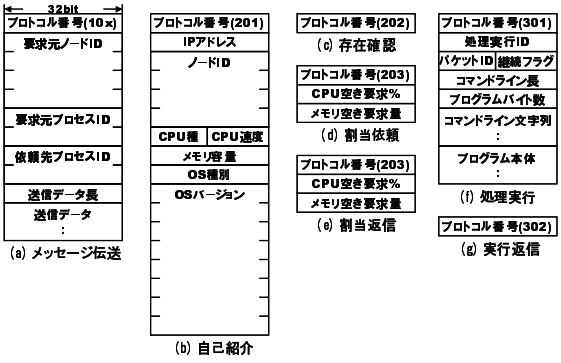

図 4: プロトコルの形式

マネージャ (CM) とアプリケーションプログラム・マネージャ(APM) (共に後述) を用意する。アプリケーションは、利用者が実行を依頼するプログラム (汎用 OS 上で普通に実行できるもの) である。

次に、メモリマップを図 3 右に示す。ページングを採用し、論理アドレス空間にカーネル、プロセス、アプリケーションが順に配置され、物理メモリにマッピングされる。ただし、ページアウト、ページインによる仮想記憶は現在のところサポートしていない。各部の領域を明確に分離し、プロセスやアプリケーションが他の領域にアクセスできないようにする。なおプロセスとアプリケーションは各々独立した論理アドレス空間を持つ。

4.2 実装の詳細

4.2.1 グループの形成と維持管理

- プロセス間通信のネットワーク対応
  各ノードが自律的に動作して最適な分散処理環境を形成する過程では、ノード間の通信が多発する。プロセス間通信機構をネットワークに対応させることで、プロセス間通信を使って他ノードと通信できるようにした (プロトコル形式:図 4 a)。

- コンパニオン・マネージャ
  コンパニオン・マネージャ (CM) はグループの生成・維持管理と処理の割り当てを行うプロセスである。グループ関連の処理は、各ノードの CM が専用プロトコル (図 4 b ～ g) で通信することで行う。

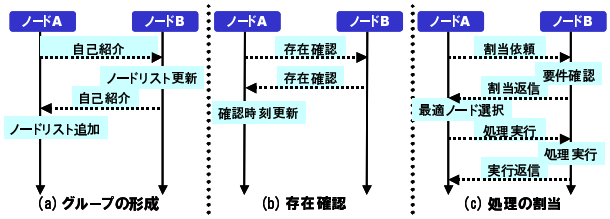

図 5: グループ管理のフロー

  - グループの生成・維持
    グループ生成の流れを図 5 a に、維持管理 (存在確認) の流れを図 5 b に示す。
  - タスク割当先の選択および実行依頼
    タスク割当先の選択からアプリケーションの実行依頼までの流れを図 5 c に示す。
    依頼元ノードは、CPU 負荷、空きメモリ容量、CPU 速度、ネットワーク速度を取得し、各要素に重みづけ合算して値の高い方から割当先ノードを選択する。重みづけの係数は、利用者が実行を依頼する時に指定可能である。

4.2.2 仮想マシン

- Linux 用システムコールの処理
  SiON は、独自のシステムコールのために割込みベクトル番号 0x40 を使用する。これはプロセスの生成・終了・待機およびメッセージ送信・受信をサポートする。さらに、Linux 用プログラムのシステムコールを処理できるように 0x80 番の割込みをハンドリングする。そのコール内容はカーネル内でメッセージに変換されて後述のアプリケーションプログラム・マネージャに送付される。

- アプリケーションプログラム・マネージャ
  アプリケーションプログラム・マネージャ (APM) はプログラムの実行を管理するプロセスである。実際にアプリケーションが実行に移されるまでの流れを図 6 に示す。

4.2.3 簡素化と高速化

- カーネルの簡略化
  SiON ではオーバーヘッド低減のためスケジューラには優先度なしラウンド・ロビンを用

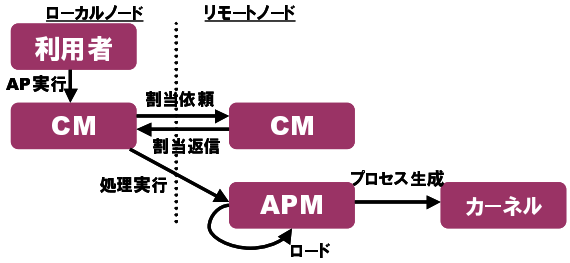

図 6: APM 処理フロー

図 7: メッセージ形式とメッセージ転送模式図

いた。また管理するデバイスを絞り込み、その処理をシンプルにして高速化した。

- プロセス間通信の簡素化
  プロセス間通信には、メッセージ・パッシング方式を用いる。プロセスはメッセージ情報 (図 7 左) を作成し、システムコールを実行する。

  カーネル領域には全物理メモリがマッピングされ、メッセージ中のデータ先頭アドレスをカーネル内の対応するアドレスに変換する。その後、送信側 (A) から受信側 (B) へ直接メモリコピーを行う (図 7 右)。

5 予備性能評価

SiON と Linux での性能評価を行なった。評価には評価には以下のスペックのマシンを使用した。また、5 回計測してその平均値を評価値とした。

CPU = Pentium II (266MHz)
MEM = 192MB
OS = SiON / Linux(Debian 3.0r0, kernel 2.4.18)

5.1 処理時間の計測

n 番目の素数を求める処理を用意し、その実行時間を計測した。(Linux では time コマンドで時間計測、SiON ではタイマ割込み回数から算出)。計測結果を表 2 および図 8 に示す。

表 2: 数値演算処理速度比較

N	SiON(sec)	Linux(sec)	高速化率 (%)
1	0.000	0.002	-
100	0.000	0.009	-
500	0.120	0.129	107.5
1000	0.500	0.560	112.0
2000	2.220	2.443	110.0
3000	5.280	5.792	109.7
5000	15.620	17.132	109.7
7000	31.840	34.916	109.7
10000	67.640	74.164	109.6
50000	1990.360	2180.970	109.6

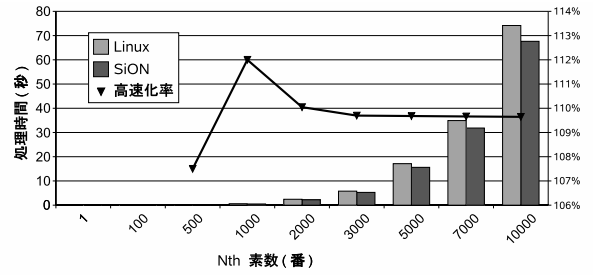

図 8: 数値演算処理速度比較

結果より、SiON のプロトタイプは Linux に比べて約 9.5%高速であることが示された。以下、SiON の方が高速である理由を考察する。

- タイマ割込処理の複雑さ
  Linux のタイマ割込処理は実経過時間の他に、プロセス毎の経過時間とインターバルタイマの更新、シグナル処理が行なわれる。これに対し SiON はシステム時間とプロセス開始時刻の更新のみ行なう。

- スケジューラの複雑さ
  Linux ではスケジューラは優先度チェックおよび優先度変更、タイムクォンタムに関わる処理を行なう。また停止できないプロセスがあるためスケジュールのオーバーヘッドが大きい。これに対し SiON ではスケジューラが単一優先度のラウンドロビンである。

- その他
  プログラム起動のオーバーヘッドは $n = 1$ の場合の結果から無視できる。また、Linux 上では必要最低限のプロセスが動作しているが、これは必要負荷である。

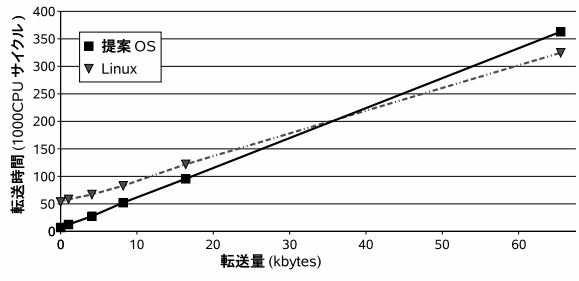

図 9: プロセス間通信性能グラフ

以上の考察から、カーネルの単純化により性能向上を図ることができたと言える。欠点は、汎用 OS として必要ないくつかの機能 (シグナルやマルチプロセスなど) を提供していないため、利用者が通常使う OS として機能不十分なことである。しかし高速化により計算能力を必要とするアプリケーションに対しては処理時間の面で有利である。

5.2 プロセス間通信の性能評価

次に、プロセス間通信によるバイト列転送の性能を評価した。Linux ではドメイン AF_UNIX の UDP ソケットを用いた。転送バイト数に対する処理時間を CPU サイクル数で計測した結果を表 3、そのグラフを図 9 にそれぞれ示す。

表 3: プロセス間通信性能比較

転送量 (bytes)	SiON(cycle)	Linux(cycle)
4	7,066	53,565
1024	12,486	58,257
4096	27,443	67,046
8192	52,158	82,901
16384	95,540	121,920
65500	362,907	324,714

結果より、SiON ではメモリ転送以外の通信準備処理が Linux の約 7.5 倍高速であること、メモリ転送自体は Linux の方が 30〜50%高速であることがわかった。この理由としては、

- 通信準備処理
  Linux ではソケットの作成や接続などに多くの処理時間を要するが、SiON ではシンプルなメッセージ作成のみである。

- 転送処理
  SiON では転送処理が mov 命令の繰返しに変換されているので、ストリング操作命令や MMX 拡張命令への置き換えにより転送速度を改善できると思われる。

が考えられる。SiON は抽象的・汎用的な通信路の提供が難しい反面、非常に高速な転送を提供しているので、多数のプロセスとの通信では通信準備の負荷が低く有利である。

6 おわりに

本稿では、計算資源を共有する一連の計算機群について、各ノードの構成が動的に変化する場合 (計算機の稼働状態の変化、ノードの参加・離脱の頻発) において、その構成変更に柔軟に適応できること、逐次最適なタスク割当先の選択を行なえること、さらに既存のアプリケーション・プログラムを実行できることを目的とした OS、SiON を提案した。またそのアーキテクチャと構成を概説したあとプロトタイプによってその有効性の検証を行った。

今後は複数のマシン上で実際に動作させ、性能やスケーラビリティなどの評価を行なっていきたい。また、実運用のためには共有ライブラリへの対応、並列処理が必要と考えられるため、その実現も今後の課題としたい。

参考文献

[1] “The Globus Project”, http://www.globus.org/
[2] “The Condor Project”, http://www.cs.wisc.edu/condor/
[3] The UNICORE Forum, “UNICORE”, http://www.unicore.de/
[4] “PC Cluster Consortium”, http://www.pccluster.org/
[5] Scyld Computing Corporation, “Scyld Beowulf”, http://www.scyld.com/page/products/beowulf/
[6] Andrew S. Tanenbaum, “Distributed Operating System”, Prentice Hall, Inc. ,1994
[7] Vrije Universiteit, “AMOEBA Distributed Operating System”, http://www.cs.vu.nl/pub/amoeba/
[8] Rob Pike, Dave Presotto, “Plan9 from Bell Labs”, http://http://cm.bell-labs.com/sys/doc/9.html
[9] 株式会社情報科学研究所, “SSS-PC 次世代オペレーティングシステム”, http://www.ssspc.org/ssspc/index-j.html