Vol. 20 No. 4　　情　報　処　理　　Apr. 1979

解　説

分散処理用オペレーティングシステムの課題†

高　平　　叡††

1. まえがき

分散処理をオペレーティングシステム (OS) の立場から見た場合にも，いわゆる通信網を介した計算処理パワーの分散，あるいはデータベースの分散，一つの計算機システム構成法としての分散，および特殊な用途を持つアレイプロセッサのような並列計算機の場合の，それぞれに目的に応じた OS の特殊性が存在する．通信網に関する主課題は，ネットワークプロトコル，データ記述言語，および共同利用資源の管理である．この OS は，多くの場合，ネットワークプログラムとして切り出した部分で，自計算機内の OS と外の世界との整合をとる．そしてユーザには，あたかも網が存在していないかのように処理すること，すなわち，分散処理を透明にすることを理想にする．そこで，プロトコルの標準化の問題とともに，どこでどのような情報や処理機能が存在するかということや，相手計算機のデータ表現やファイルコンベンションの違いを吸収する方法など，網管理上の問題がでてくる．

並列計算機の場合は，文字通り並列処理そのものが課題であり，計算対象の並列処理への分解，単位プロセッサへの処理の分散と統合である．

計算機システムの構成法としての，すなわち計算機複合体の，分散処理は，それぞれの計算機アーキテクチャに添った OS を構成することと共に，前二者と共通する課題を持っている．たとえば，ネットワークプロトコルの高位レベルは，網の存在如何にかかわらず，共通化しようと考える．それは，ユーザの立場から計算機の使い方を同一にしたいという要求であると共に，システムにとっては，規模や機能の拡張を容易にする手段になる．また並列計算機における同期処理をはじめ，共通する技術を持っている．

分散処理とよばれる分野は広く，各分野ごとにそれぞれ問題を持ってはいるが，本稿ではすべてにふれる余裕は無いので，通信網および並列計算の問題に直接ふれることはせず，計算機複合体の OS を中心に，その技術と課題について述べる．

技術自身は若く，多くの部分は現在個々のシステムで研究開発が進められている段階にあるので，これを一般的に議論するには困難さがあるが，実現例を引き合いにしながら述べる．

2. 分散処理 OS における問題点

分散処理をする目的は

- 単位の計算機またはプロセッサの数を増減することによって得られるシステムの処理能力と機能の拡張性
- 並列処理による高速処理
- 多数のプロセッサの相補による高信頼性
- 廉価な小型の計算機の利用による良好な価格性能比
- 資源の共同利用

などである．OS を設計するときは，これらの目的を十分生かす工夫をしなければならない．

目的に応じて多数台の計算機またはプロセッサを組合せ，一つのシステムを構成するとき，個々の計算機またはプロセッサに，どのような仕事の与え方をするか，すなわち，処理を分散するかによって OS の形態が違ってくる．分散の種類は観点によって種々の分け方ができるが[1]，OS に関連づけると，高速演算処理を目標にしたデータの分散，ユーザ対応処理の分散，負荷の分散，従来の OS の内容を機能対応に分けた機能分散などの形がある．ここでは，代表的な負荷分散と機能分散の場合について考えていく．

つぎに，分散処理の OS に特徴的ないくつかの課題をあげる．

(1) 結　合

計算機またはプロセッサ間の結合が，ハードウェア

† A Few Remarks on Operating System for Distributed Processing by Satoshi TAKAHIRA (Yokosuka Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話公社横須賀電気通信研究所

的にでき上がった後の，処理の実行主体であるプロセス間の結合において考えるべきことは

1) 通信の方法，規約
2) プロセス間の処理の同期
3) 共通資源の管理情報の管理
4) 異なるファイル系へのアクセス処理

などである．3) は網レベルの分散処理の場合にも，4) は異機種の計算機の結合の場合にも特に共通する事項である．

結合については，システムの拡張性と信頼性に留意しなければならないので，通信の規約，すなわちプロトコルの規定が重要になる．

（2） 性　能

n 台のプロセッサから構成されるシステムの最大能力が n 台にし得ない理由は，分散のやり方によって多少異なるけれども，次のような原因による．

1) 共有資源の使用時のアクセスの競合
2) シリアルなアルゴリズムを並列なアルゴリズムに分解し，処理した後統合するためのオーバヘッド
3) 並列処理されるものが，必ずしも同型でないために各プロセッサに生じる処理の空

これらの各々をどのように小さくおさめるかが，アーキテクチャおよび OS の課題である．

（3） 負荷のバランス

性能を制限する理由の一つに，並列処理されるものが同形でないために生じるプロセッサの空をあげた．この問題は，機能分散型のシステムにおいて顕著な問題である．**図-1** は OS の機能を 6 つに分割した機能分散型のシステムで，通常の TSS ユーザプログラムの負荷が，6 つのプロセッサクラスにどのように配分されるかを調べた測定結果の例である[2]．この図から二つのことが観察される．一つは，負荷がプロセッサクラスの間で大きくかたよっていることであり，一つは，ユーザプログラムが変わると，負荷のかたよりの形が変ってくることである．後者は見方を変えると，TSS においては負荷のバランスが時間的にも変動するということになる．負荷をバランスさせるためには，機能の分散のさせ方をさらに工夫しなければならないが，分散した一つの機能のまとまりも必要であり，非常に難かしい問題を含んでいる．時間的な変動に追従するような対策も場合によって必要になる．

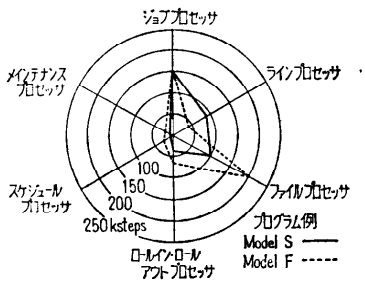

図-1 プロセッサクラスごとの負荷の分布

（4） システムの再構成

多数のプロセッサを相互矛盾なく立ち上げること，故障した装置をシステムから切り離したり，修理の終った装置をシステムに組み入れる場合に，他の処理に悪い影響を及ぼさないようにしながら処置すること，さらには，システムが故障したとき，複数のプロセッサにまたがって処理されていたジョブを救済すること等，障害処理まわりの問題がある．システム管理の一元化あるいは多元化，障害時の通信の方法等システムの信頼性ばかりでなくシステムの拡張性にもつながる．

（5） OS の生産性

分散処理の OS の機能的な仕様の定まった後の，いわゆる作成面についても，多くの問題が提起される．複数のプロセッサがそれぞれ別の OS を持つ場合にはそれらを能率よく作成したい．分散処理の各要素のプロセッサの機能を追加したり変更したりすることが容易なように OS を構成したい．

一方，OS の生産性に注目して別の問題がある．それは，分散処理の特徴の中に，分散することによってソフトを作り易くし得るということである．根拠は，従来の OS を分割分散すると，個々の OS の規模は小さくできるため，あるいは，廉価なハードウェアを多数使って分散を進める一方で，OS をマルチプロセッシング機能などの無い簡単なものにすることができるため等である．このようなことの可能性も考えに入れなければならない問題である．

3. OS 設計における諸考察要因

前節に述べたような問題を具体的な OS の設計において考えていくとき，選ばなければならないいくつかの考察要因についてふれる．

（1） OS の構成——実行管理のおき方

多数台のプロセッサの処理実行制御の，プロセッサ相互の関係のとり方には，大別すると次の三つの構成の仕方があり，それぞれに得失がある[3]．

1) マスタ／スレーブ方式

マスタとなるプロセッサが固定していて，それが他のスレーブプロセッサに，仕事を分配する構成のものである．構成は比較的簡単であり，単一プロセッサが処理の実行を制御するので，テーブルの参照のロックアウトの解決が容易である．スレーブプロセッサにあまり能力を持たせない場合に有効である．短所は，マスタプロセッサがダウンすると，直ちにシステム全体もダウンしてしまう可能性があることである．各プロセッサの仕事量がよく把握できているような特定の応用に向いた構成である．例えばミニコンとマイクロプロセッサからなる画像処理用のシステムなどは条件に合う．

2) 各プロセッサが，それぞれ実行管理を持つ方式

各プロセッサは，自分の中に出て来た要求を自分自身で処理する構成のものである．各プロセッサは，主に，自分のプライベートなテーブル上で仕事を処理するが，一部分は，システム共通なテーブルが必要であり，それに対するアクセスの制御を要する．マスタ／スレーブ方式と違って，個々のプロセッサの障害がシステムに及ぼす影響は小さくすることができるが，一方，修理後のプロセッサの組入れには難しさのでてくる可能性がある．本方式は，各プロセッサが I/O やファイルを持った計算機の場合の結合にも適用できる．この構成をとっている例には，PRIME[4] や Tandem 16[5] などがある．

3) 対称型の構成

システムのすべてのプロセッサを他の資源と同じように対称的に，あるいは一つにプールして使う．マスタプロセッサは固定しておらず，処理の流れに従って次々に移っていく．こうすることによって負荷のバランスを良く保つことができる．サービス要求の競合は，優先順位づけをすることで解決できる．実行管理のプログラムは，同時にいくつかのプロセッサで実行されることになるので，リエントラントに作る必要がある．この構成は最も難かしいものになる．

（2） プロセッサ間通信

プロセッサ間の通信については，通信のプロトコルと，通信処理によるオーバヘッドの影響について考慮する必要がある．

通信のプロトコルの規定は，通信網のレベルでは，国際的な標準化活動が精力的に進められているが，ユーザ側に近い OS あるいは応用プログラムの階層の高位レベルのものについては，当面自分で解決して行かなければならないし，あるいは，個々のシステムの特徴を生かすためには，それぞれの問題とも言える．

プロセッサあるいは処理の単位としてのプロセス間の通信には，同期と転送という二つの機能が必要である．同期をとるためには Dijkstra の semaphore の概念や，Test and Set 命令などが用いられる．情報の転送には，プロセッサ間に論理的なリンクを構成した後にデータを転送するバーチャルコール型のものと，リンクを構成せずにデータメッセージを転送するメッセージ通信型（あるいはデータグラム）のものが代表的である．前者は多量のメッセージ転送に，後者は小量の転送に適している．具体的な例をあげれば，ACE[6] では，ローカルメモリを他モジュールからアクセスさせるモジュール間通信の機能と共用資源に対する相互排斥モジュールを用いて OS 間の通信を行っている．MICS II[7] では，多階層のプロトコルを設け，メッセージ通信型の通信機能を実現している．KOCOS[8] での通信は，メッセージ通信型のものであるが，ランデブーテーブル上で，送受の要求のつき合せを行う方法をとっている．

通信の形には，一対一の通信の他に，一対多の通信の形がある．これを一斉通信あるいはブロードキャストなどと呼んでいるが，分散処理システムの中では特異なはたらきを持っている．10^5〜10^6 個のプロセッサを結合できるという CHOPP[9] では，ブロードキャストの機能を用いて，処理の要求を伝えている．処理を要求するプロセッサは，自分のアドレスを全プロセッサに向ってブロードキャストする．もし，その処理を受けることのできるプロセッサが存在すると，そのプロセッサは自分のアドレスを要求プロセッサに返答する．要求プロセッサは，返答の中から一つを選び，処理の要求情報を選んだプロセッサに伝えると共に，他のプロセッサには，要求を取消すことをブロードキャストする．

ブロードキャストは，後に述べるように，強制通信機能と合わせ，保守や障害処理にも使われている．

次に，通信のオーバヘッドを少なくする工夫として，転送する情報量を見きわめ，バーチャルコールとメッセージ通信を使い分ける．また，通信の発生頻度と転送情報量に応じて，個々のデータを区分し，情報の転送ルートを変えようという工夫もある[10]．図-2 はプロセッサ間で交わされる通信の情報量と発生頻度を観測した例であるが，比較的情報量は小さいが発生頻度の多いメッセージタイプのデータと，情報量の多いデ

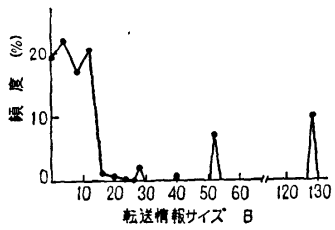

図-2 転送情報の分布

ータタイプに分かれている[11]．そこで，メッセージ型のデータを高速バスで，情報量の多いデータはメモリ経由で転送させている．

（3） 共通メモリ

マルチプロセッサシステムにおける大きな問題は，メモリ参照の競合による能率の低下である．そこで，分散処理システムの多くは，各プロセッサがそれぞれプライベートなメモリを持ち，極力自メモリ内で仕事を進められるようにアーキテクチャ上で配慮している．しかし，システム共通情報は依然として存在するので共通メモリを設けることになるが，OS 構成上は，アクセスを少なくする工夫がいる．

これに反して，共通メモリを廃し，共通情報として必要な情報は，通信を介して受け渡しする方法を HXDP ではとっている[12]．これは，情報が共通に使われ頻度は比較的少ないという判断と，処理の対象を固定するという条件から来ているが，処理対象の動作を十分解析して設計する必要がある．

共通メモリの切り出しや返却の制御は一元的に行わないと管理が困難になるので，そのために専用のプロセッサ OS を設けているものもある．

（4） 負荷の平滑化

機能分散型のシステムでは，特定のプロセッサに負荷が集中し，そのプロセッサをボトルネックにして，全体の処理が進まなくなることが生じうる．このために，OS には，あらかじめ負荷の状態を観測し，過剰な負荷を他にまわすか，あるいは抑制する管理が必要になる[13]．このためには，観測点，判断の規準，判断後のフィードバックの方法等，詳細な OS の解析がなければならない．

アクティブな平滑化の方法として，重負荷のプロセッサの負荷を，軽負荷のプロセッサにまわすことを可能にするために，プロセッサの OS とファームウェアをダイナミックに入れかえてしまう方法も提案されている[1]．

（5） 障害処理

アーキテクチャ上から高信頼性を目的としたシステムに Tandem 16 がある．このシステムでは，システムの構成要素をすべて多重系のバスで結合し，一部に障害があっても動作可能になっている．各プロセッサは，OS のコピーを持っている．周期的にソフトウェアによるチェックが行われ，故障が発見されると自動的に切り離なされる．

このような高信頼化指向のシステムを別にしても，分散処理の場合には，信頼性に対して多数台であることを積極的に利用すべきである．多数台の利用には，プロセッサ相互の動作状態のモニタリング，相互診断，故障時の代行処理などがあげられる．

OS 構成上からは，負荷分散型のシステムでは，Tandem 16 の例のように，故障に対して負荷を分担するプロセッサが減少するだけに止められるが，機能分散型のシステムでは，故障したプロセッサの機能がシステムから欠落してしまうことになるので，システムダウンにつながる．これを避けるために，システム構成上，同一機能を多重化しておくか，または，他のプロセッサが直ちに肩がわりして処理を続行するような対策がいる[14]．

（6） モジュール化

OS をモジュールと呼ぶ比較的小さな単位に分け，それ等を積み上げてシステム全体を構成しようということは，OS の作成と保守を容易にするために，長年研究されて来た．

分散処理の場合にこれが特に意識されるのは，多数のプロセッサの OS を作成し保守しなければならないからである．もちろん，すべて一つの OS のコピーですむ場合もあるし，既成の OS の一部改造ですむ場合もある．OS の機能を分散したシステムでは，多数のハードウェア上に OS を分割するためにモジュール化

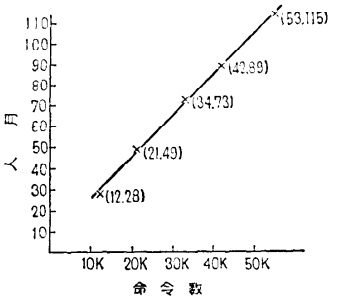

図-3 プログラムの開発工数 (Hydra)[15]

を進める必然性も出てくる.

モジュール化の効果を示すものとして, C. mmp の OS である Hydra での実験がある. ソフトに新たな機能を付け加えるときに必要な努力は, システムの規模が増大するにつれ増大する, という傾向に対して, モジュール化して作成した Hydra では, 規模の大きさには無関係にすることができた (図-3)[15].

モジュール化は, 通信制御機能やファイル制御機能などのサービス対応の機能で分けるものと, ハードウェアを制御する部分と OS の中核部, OS のユーザ対応部といった階層に分けるもの, および, この二つの分け方を組合せたもの等がある. Hydra のモジュールの概念は, もっと厳密に定義されたものであるが, OS を kernel と呼ばれる部分と, ファイルやコマンド言語などの non-kernel な部分に分け, それぞれを非階層的なモジュールに分解している.

4. むすび

以上, 分散処理の OS に特徴的ないくつかの問題を概観したが, この他にも, ふれなかった通信網のプロトコルまわり, データ記述言語, あるいはセキュリティ等に関し, それぞれに問題がある.

OS はシステムの目的にそって設計するものであるから, 処理の対象をよく把握した上で, ここに述べたような問題に対して, それぞれ方法を見出していかなければならない.

そして, OS は, システムの能力を最大限引き出すように作られていくのであるが, ここで視点をシステム作りという位置に移すと, 先にも述べたように, 分散処理の重要なあるいは最大といってもよい課題は, 超 LSI に期待される安価なハードを十分使い, ソフトウェアを作りやすくすることであろう.

分散処理の OS の技術はまだこれからのものであり, 解決しなければならない問題が山積しているが, それらに対する努力は, 分散処理に期待される効果によってむくわれよう.

最後に, 引用した例が, 筆者の関連した PPS に偏したことをお詫びする.

参考文献

1) 相磯秀夫: 機能分散型計算機システム, 情報処理, Vol. 18, No. 4, pp. 325-333 (1977).
2) Murakami, K. et al.: Poly-Processor System Analysis and Design, Proc. of 4th Ann. Symp. on Computer Architecture, pp. 49-56 (1977).
3) Enslow, P. H.: Multiprocessor Organization A Survey Computing Survey, Vol. 9, No. 1, pp. 103-129 (1977).
4) Baskin, H. B.: PRIME: A moduler architecture for terminal oriented system, Proc. SJCC, pp. 431-437 (1972).
5) Tandem 社 Tandem 16 system introduction.
6) 藤井, 飯塚, 古谷: ACE 11 システムとその基本 OS, 信学技報, Vol. 76, No. 216, pp. 25-34 (1977).
7) 山崎, 他: マルチプロセッサシステム MICS II のシステムコントロール, 信学技報, Vol. 76, No. 216, pp. 35～44 (1977).
8) 上林, 他: ミニコンピュータ複合システム KOCOS のプロセッス間通信機能, 信学会論文誌, Vol. 61- D No. 3, pp. 186-193 (1978).
9) Sullivan, H.: A Large Scale, Homogenous, Fully Distributed Parallel Machine, II, Proc. of 4th Ann. Symp. on Computer Architecture, pp. 118-124 (1977).
10) Nishikawa, S. et al.: Interconnection Unit for Poly-Processor System: Analysis and Design, Proc. of 5th Ann. Symp. on Computer Architecture, pp. 216-222 (1978).
11) 村上, 佐藤, 小山: ソフトウェアシミュレーションによるポリプロセッサシステムの動作解析, 信学会論文誌, Vol. 61-D No. 2, pp. 119-126 (1978).
12) Jensen, E. D.: The Honeywell Experimental Distributed Processor-An Overview COMPUTER, pp. 28-37 (Jan. 1978).
13) 村上, 高橋: 機能分散システムにおける負荷管理方式, 昭和 52 年度信学会情報部門全国大会, 347, p. 347 (1977).
14) Takahira, S. et al.: A Reliability Aspect of Function Distribution System: PPS proc. 3rd UJCC, pp. 70-74 (1978).
15) Wulf, W. A. et al.: Overview of the Hydra Operating System Development. proc. of 5th Symposium on Operating System Principles, pp. 122-131 (1975).

(昭和54年1月8日受付)