オペレーティング・システム　55－1
（１９９２．６．８）

動的負荷分散におけるメッセージ交換の削減

芦原　評

電気通信大学　情報工学科

ジョブの到着・終了時のノード間移送による動的負荷分散は、分散システムの性能を大きく向上させるが、十分な効果を得るためにはノード間での大量の情報交換が必要となり、そのためのコストが負荷分散の効果を減殺しかねない。ここではジョブの移送時の問い合せメッセージを制限するためのノード情報を、問い合せメッセージ自体によって維持するアルゴリズムを与え、シミュレーションによって評価した。その結果、同アルゴリズムが負荷分散の完全性を減ずることなく情報交換のコストを低下させることが示された。さらに、中央制御ノードが利用可能であれば、メッセージの総数は大幅に減少する。

REDUCTION OF INFORMATION EXCHANGE FOR ADAPTIVE LOAD SHARING IN DISTRIBUTED SYSTEMS

Hyo Ashihara

Department of Computer Science and Information Mathematics
University of Electro-Communications

Algorithms to reduce the number of messages to probe nodes for adaptive load sharing in distributed systems are proposed, evaluated and compared using simulation. Adaptive load sharing by job migration improves the performance of distributed systems greatly. However, it can require much cost for information exchange among the nodes. It is shown that use of load table which is updated by the probing itself can reduce the number of probing messages effectively, while the completeness of load sharing is not lost. In addition, if a central control node is available, centralized algorithms can further reduce the number of messages.

1．はじめに

多数のノードからなる分散システムにおいて、特定のノードに負荷が集中することを防ぐ負荷分散により、その性能を飛躍的に向上させることができる[11]。特に、各瞬間における負荷の変動に対応する動的負荷分散は、各ノードの平均負荷のみを均衡させる静的負荷分散に比べても大きな効果を持ち[8]、多くの研究が行なわれてきた。しかし、このためにはシステムの動的な状態を常に把握する必要があり、そのためのメッセージ交換があまりに大量であれば、性能に影響する恐れがある[5]。不完全な情報、あるいは最も極端な場合自ノードの状態のみに基づいて移送を決定する場合[3,5,7]には相応の性能低下を伴う。

動的負荷分散の場合、負荷の「均衡」[10]よりは無駄な空ノードの発生（システム内での、複数のジョブを持つノード――以下、重ノードと呼び、「ノードが重である」のようにも使う――とジョブを持たないノード――空ノード――の混在）を防止することが本質的に重要である[3]。従って、

1．現在空でないノードに外部から新たなジョブが到着した場合、システム内に現在空のノードがあれば、その一つにジョブを移送する（送り手主導移送）。
2．現在ジョブを一つしか持たないノードにおいてその唯一のジョブが終了する場合、システム内に複数のジョブを処理中のノードがあれば、その一つからジョブを移送する（受け手主導移送）。

の２つの操作を行なえばよい[1]。

おのおのの移送を完全に行なうためには、移送の主導ノードはシステムの他のすべてのノードの現在の状態を知らなければならない。

移送のたびに相手ノードに問い合せメッセージを送りその返答を受ける方式[2-4]では、候補となるノード数が大きい場合メッセージの総数が非常に多くなる。移送の開始はメッセージの交換後に行なわれるため、メッセージ交換の遅れはただちに移送の遅れとなり移送の効率を低下させる。状態情報の最新性は保証され状態変化に対する追随性も良いが、問い合せメッセージによって得られた情報は毎回破棄されるため無駄が大きい。

一方、各ノードがあらかじめメッセージを送り合い他のノードの状態を負荷テーブルに記憶しておく方式[6,10]では、すでに変化した可能性のある過去の情報に基づいて移送が決定されるため、効率が落ちるだけでなく、システム全体の不安定化[2,3]を招く恐れすらある。

両者の融合型として、過去の情報に基づいて移送の対象をある程度絞った上で、それらに対し改めて問い合せメッセージを送るという方式が考えられる。その場合も、負荷テーブル[9]の維持方法が問題となる。定期的な情報交換によった場合、その頻度を増やせば無駄なメッセージが増加し、減らせば情報の精度が落ちる。（最終的な問い合せを行なうことで不適切なノードを移送相手に選ぶことは避けられるが、適切なノードを見落す恐れがある。）またテーブルの更新と最終的な問い合せのために二重にメッセージを送る必要がある。

本稿においては、問い合せメッセージ自体によって上記テーブルの更新を行なう一方式を提案し、その評価を行なう。この場合、最終的な情報は問い合せによって得られるためその時点で最新であり、また移送が必要となった時に交換が行なわれるため無駄な情報が削減できるが、常に必要なノードにメッセージが送られることを保証しなければならない。シミュレーションの結果、負荷分散に必要な情報の精度を落とすことなく、不要な情報交換を大幅に削減できることが示された。

また、中央制御ノードを導入することが可能ならば、それによってさらにメッセージ数を減らすことができる。

以下、第２項で分散システムのモデルを、第３項で情報交換とジョブ移送のアルゴリズムを、第４項でシミュレーションによる評価を述べた後、第５項で結論を簡単にまとめる。

2．システムのモデル

ここでは、N個のノードからなる以下のような分散システムを考える。（[4]と類似である。）

各ノードは同一性能の単一プロセッサを持つ。相互に独立した、プロセッサ依存、単一タイプのジョブが各ノードに同一のポアソン分布をもって到着する。ジョブの処理に要する時間は指数分布とする。各ノードにおけるジョブ処理はFCFSでもラウンドロビン型でもよい。すなわち、負荷分散を行なわない状態で、各ノードはM/M/1待ち行列を構成する。システム負荷（ジョブ平均到着率×平均処理時間）をρとする。

すべてのノード間は高速のネットワークによって接続されている。ただしブロードキャストは仮定しない。任意のノード間で、ジョブを移送することができる。ジョブ移送に要するコストは送り手側プロセッサの負荷として表現されるとする。ジョブ移送は通常のジョブ処理より高い優先度を持ち、同時には一つのジョブしか移送できない。すなわち、ジョブの移送が開始されてから終了するまで、送り手側ノードではジョブ処理を中断する。受け手側では、移送と同時にジョブ処理を開始できる。ジョブ移送に必要な時間は指数分布し、どのノード間でも等しい。平均移送時間／平均処理時間をτとする。τは十分小さい。また、各ジョブの処理時間と移送時間は独立である。

状態情報メッセージについてはコスト・遅れは考えず、総数のみを評価する。各メッセージはただちに到着し処理され、ジョブ処理・移送自体には影響しないものとする。

以上の仮定に基づくモデルは極めて単純なものではあるが、本稿の目的は特定アルゴリズムの厳密な性能の測定ではなく、アルゴリズム相互の基礎的な比較にあるため、仮定が中立的である限り第一次的な評価としては十分である。メッセージ総数は情報交換コストの尺度となり、逆にメッセージ数が十分制限できるならばコストを無視した仮定が現実性を持つ。

3．情報交換アルゴリズム

以下、単純逐次（NS）、単純並行（NP）、分散逐次（DS）、分散並行（DP）、集中逐次（CS）および集中並行（CP）と呼ぶ６つの情報交換（ノード探索）アルゴリズムを示す。

いずれのアルゴリズムにおいても、送り手主導・受け手主導の２種のジョブ移送が行なわれる。どちらの移送においても、移送は完全である。すなわち、空ノードが重ノードを、重ノードが空ノードを、それぞれの発生時に発見し損うことはない。従って、ジョブ移送によるシステム自体の性能はすべてのアルゴリズムで同一である。

ただし、モデルの項で述べたように、一つのノードは一時には一つのジョブしか移送できない。このために、結果的にはシステム内に重ノードと空ノードの混在が発生する。複数ジョブの移送を許すか、ジョブ移送終了時に次の移送を開始することとすればこれを避けることができ、性能が向上するだけでなく、重ノードと空ノードが同時には存在しないという知識を利用すればアルゴリズムの簡略化が可能となる。

しかしながら現実のシステムにおいても一時的に移送が不可能な状態は常に発生しうるものであり、このような特殊性を前提とした最適化は好ましくないであろう。

3．1．逐次探索と並行探索

ブロードキャストを仮定しないため、複数ノードに対する問い合せは各個に行なわねばならない。そのための方法は性能に影響するが、どのようなものが許されるかは、システムにも依存する。ここでは、以下の二種を比較する。

逐次探索とは、一つのノードに対する問い合せへの返答を受取った後、次のノードへの問い合せを行なう方式である。問い合せの順序はランダムに決せられる。現在の問題では、返答をYes/Noに限った時、Yesであればそれ以上の問い合せを行

なわなくてよい。従って、Yesの返答を得られる可能性が高い時は有効な方法であるが、最悪ではすべての候補ノードに問い合せねばならない。

並行探索とは、すべての候補ノードに対し同時に問い合せメッセージを送る方式である。メッセージを受取ったノードのうち、返答がYesであるもののみが返答する。並列性と、Noの返答が省略できることが利点である。以下の文章で、「Noの返答」は並行探索の場合「返答なし」を意味する。

3．2．単純探索

NSおよびNPアルゴリズムでは、重ノードでないノードがジョブ処理を終了した時（すなわち空ノードとなる時）他のすべてのノードに対し問い合せメッセージrecvを（逐次ないし並行に）送る。recvを受取ったノードのうち、すでに移送中でない重ノードはYesを、それ以外はNoを返答する。空ノードは、最初にYesを返したノードを送り手として選び、そこから一ジョブを移送する。

同様に、空ノードでないノードにジョブが到着した時にはメッセージsendを送り、空ノードからの返答Yesがあればそこに対しジョブを移送する。

移送相手の決定後に必要とされる各種の情報交換は移送コストに含まれるとする。

3．3．分散探索

DSおよびDPアルゴリズムにおいては、各ノードは他ノードの状態を記憶する２つのベクトル「重」「空」を持つ。ノードiの持つベクトルの第j要素は各重(i,j)および空(i,j)と記述される。各要素は1又は0の値を取り、初期値は1である。重(i,j)=0は「ノードjが重でないことをノードiが知っている」ことを、空(i,j)=0は「ノードjが空でないことをノードiが知っている」ことを、それぞれ意味する。以下にDSおよびDPのアルゴリズムを示す。なおここでi=jの場合は無視する。

ノードiが空となった時
　iはrecvを重(i,j)=1である各ノードjに送る。
　　recvを受取ったノードjにおいて、
　　　jが重でないなら、
　　　　Noを返し、空(j,i):=1,重(j,i):=0とする。
　　　　Noを受取ったiでは、重(i,j):=0,空(i,j):=1とする。
　　　jが重であって移送中でないなら、
　　　　Yesを返し、空(j,i):=0,重(j,i):=1,
　　　　重(i,j):=1（変化せず）、空(i,j):=0。
　　　jが重であって移送中なら、
　　　　No'を返し、空(j,i):=1,重(j,i):=1,
　　　　重(i,j):=1,空(i,j):=0。
　　（No'は並行探索の場合も明示的に送られる）
　iはYesの返答を返したjがあればその最初のものを送り手として選ぶ。
空でないノードiにジョブが到着した時
　ジョブの総数が2で、iが移送中でないならば、
　　iはsend1を空(i,j)=1であるノードに送る。
　　　jが空でないなら、
　　　　Noを返し、重(j,i):=1,空(j,i):=0,
　　　　空(i,j):=0,重(i,j):=1。
　　　jが空なら、
　　　　Yesを返し、重(j,i):=0,空(j,i):=1,
　　　　空(i,j):=1,重(i,j):=0。
　ジョブの総数が3以上で、移送中でないならば、
　　iはsend2を空(i,j)=1であるノードに送る。
　　以下はsend1と同様、ただし
　　　jが空の時、
　　　　重(j,i):=1,空(j,i):=1,空(i,j):=1,
　　　　重(i,j):=1。
　　　　さらに、jが受け手となった場合、
　　　　　空(i,j):=0。
　ジョブの総数が2で、iが移送中ならば、
　　iはsend3を空(i,j)=1であるすべての（逐次探索であっても）ノードjに送る。
　　　返答は必要とされず、重(j,i):=1,
　　　空(j,i):=0,空(i,j):=1,重(i,j):=1。
　iは最初にYesを返したjを受け手として選ぶ。

以上のアルゴリズムにおいて、すべてのi,jに対し常に、空(i,j)=1であるか又は重(j,i)=1であることが保証される。また空(j,i):=0となるのはiが空でない（又は、なくなる）場合のみである。ゆえに、iが空となった時、

1）現在、空(j,i)=1である
2）移送が行なわれ、iが空でなくなる
3）重(i,j)=1である。recvがjに送られるが移送は行なわれず、空(j,i):=1となる

のいずれかとなる。従って、iが空であるときは常に空(j,i)=1であり、jが空ノードiを「見落す」ことはない。

同様に、iが重ならば、すべてのjについて重(j,i)=1である。

よって、分散探索アルゴリズムは負荷分散に関し単純探索と同様に機能する。

3．4．集中探索

CSおよびCPアルゴリズムにおいては、Nノードに加えて一個の中央ノード0を想定する。中央ノードはジョブの処理を行なわない。（従って、Nノードの一つを中央ノードとすることもできる。）中央ノードはN要素からなる2つのベクトルをもち、第iノードに対応するおのおのの第i要素を重(i)、空(i)と書く。中央ノード以外の各ノードは２つのビットを持ち、ノードiのそれは重'(i)および空'(i)と書かれる。重'(i)=0ならば、ノードiはシステムに重ノードがないことを知っている。

ノードiが空になった時、重'(i)=1ならば、
　iはrecvを0に送る。
　0はrecvを重(j)=1である各jに送る。
　　jが重でないなら、
　　　jは0にNoを返し、重(j):=0,空'(j):=1。
　　jが重で移送中でないなら、
　　　jは0にYesを返し、空(j):=0,重'(j):=1。
　　jが重で移送中なら、
　　　jは0にNo'を返し、空'(j):=1,空(j):=0,
　　　重'(j):=1。
　各jから0への全返答がNoなら、
　　0はiにNoを返し、重'(i):=0,空(i):=1,
　　重(i):=0,空'(i):=1。
　Yesの返答がなく、少なくとも一つがNo'なら、
　　0はiにNo'を返し、重'(i):=1,空(i):=1,
　　重(i):=0,空'(i):=1。
　Yesの返答があれば、
　　0はiにそのノード番号と共にYesを返し、
　　空(i):=0,重(i):=0,空'(i):=1。

空でないノードiに到着があり、空'(i)=1の時、
　iのジョブの数が2で、移送中でないならば、
　　iはsend1を0に送る。
　　0はsend1を空(j)=1であるノードjに送る。
　　　jが空でないなら、
　　　　jは0にNoを返し、空(j):=0,重'(j):=1。
　　　jが空なら、
　　　　jは0にYesを返し、重(j):=0,空'(j):=1。
　　0への全返答がNoなら、
　　　0はiにNoを返し、空'(i):=0,重(i):=1,
　　　空(i):=0,重'(i):=1。
　　Yesの返答があれば、
　　　0はiにYesを返し、空'(i):=1,重(i):=0,
　　　空(i):=0,重'(i):=1。
　iのジョブの数が3以上で、移送中でない時、
　　iは0にsend2を送る。
　　以下はsend1と同様、ただし、
　　　jから0への（従って0からiへの）返答がYesの時、
　　　　重'(j):=1,空'(j):=1,
　　　　重(j):=0,空(i):=0,重(i):=1,
　　　　重'(i):=1,空'(i):=1。
　iのジョブの数が2で、移送中の時、
　　iはsend3を0に送る。
　　0はsend3を空(j)=1であるすべてのノードに送り、空(i):=0,重(i):=1,重'(i):=1,
　　重'(j):=1。返答は必要としない。

集中探索アルゴリズムも、単純および分散探索と同様の結果をもたらす。

4．シミュレーション結果

以上の6アルゴリズムについてシミュレーションを行ない、アルゴリズムの正当性を確認すると共にメッセージ（問い合せおよび返答）の数を比較した。一回のシミュレーションは，ジョブの平均処理時間の10^5倍に相当し、一つのパラメーターの組に対し10回の平均を取った。同時に得た標準偏差から見て、統計的な誤差は無視できる。

図1は負荷分散自体の性能を示す。さまざまなノード数Nに対し、負荷ρを変化させた時のジョブの平均応答（投入から終了まで）時間／平均処理時間である。τ=0.1としている。すでに述べたように、この結果は比較したアルゴリズムによらない。明らかに、性能はNの増加と共に向上するが、非常に大きなNではその効果は少ない。

図2から図4はメッセージの頻度 mを示す。すなわち、一ノード（集中探索法での中央ノードを除く）における、問い合せおよび返答メッセージの送信および受信の総数の、ジョブの平均処理時間あたりの平均である。図2ではN=64、τ=0.1のもとでρを、図3ではρ=0.8、τ=0.1のもとでNを、さらに図4ではN=64、ρ=0.8のもとでτを、それぞれ変化させている。ここに挙げた以外のさまざまの条件においても検討したが、以下に述べる全体的な傾向に変化はなかった。

単純探索アルゴリズムでは、Nとρの増加に伴いmが急激に増大する。並行探索は逐次探索よりわずかに良い。反対に、過去の情報を用いて探索の効率を上げた場合には、逐次探索の方が並行探索より少ないメッセージ数ですむ。逐次探索はまた、実装が容易であり、複数のノードから同時に問い合せを受けた場合の処理を考慮しないでよいなどの利点も持つ。ただし、実時間での総コストについては単純な比較はできない。

DSアルゴリズムでは単純探索に比べメッセージ数を大幅に削減することができるが、Nが大きい時にはなお大量のメッセージを要する。一方で非常に大きなNに対しては、性能の向上は飽和状態になるため、各ノードから見た移送相手の候補ノードをあらかじめシステムの一部に限定して負荷分散を行なう（ネットワークを完全に分割する必要はない）ことにより、性能を大きく犠牲にせずにメッセージ数を減らすことができる。分散探索アルゴリズムは、任意のネットワークトポロジーを持つシステムに応用できる。

CSアルゴリズムは、大きなNにおいてもメッセージ数を制限できる。ただし、この場合、中央ノードがボトルネックとなる。中央ノードは、他の全ノードの合計と等しい数のメッセージを扱わねばならない。中央ノードが過負荷の場合、システムを小ネットワークに分割することができる。

図4からわかるように、ジョブ移送のコストτは決定的な要因とはならない。特に、負荷分散自体が有効性を持つのはτが小さい場合であり、この時τの変化の影響は少ない。

5．おわりに

分散システムのノード間でジョブを到着・終了に従って移送し、空きノードの発生による計算能力の浪費を防ぐ動的負荷分散は、システムの性能を大きく向上させるが、十分な効果を得るためにはノード間での大量の情報交換が必要となる。ときにはそのコストをおさえるために情報の質を犠牲にする必要が生ずることもある。しかし、ごく単純な情報交換法と比較した場合、その無駄な部分のみを削減することが可能である。

本稿では、ジョブの移送時の問い合せメッセージを制限するためのノード情報を、問い合せメッセージ自体によって維持するアルゴリズムを与え、評価した。シミュレーションは、同アルゴリズムが負荷分散の効果を減ずることなく情報交換のコストを低下させることを示した。さらに、中央制御ノードが利用可能であれば、メッセージの総数は劇的に減少する。

無論、これはそのような条件を満たす唯一のアルゴリズムではないし最適のものでもないが、動的負荷分散機構の実用化をめざす一つの試みとして、今後の研究への一助となれば幸いである。

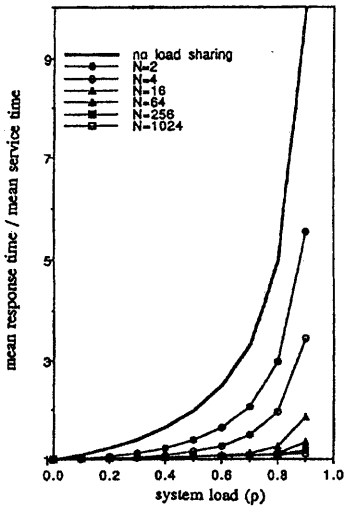

図1　動的負荷分散による性能向上

$\tau = 0.1$

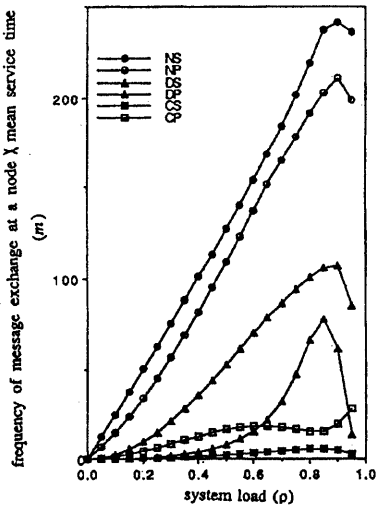

図2　システム負荷に対する情報交換の頻度

$N = 64, \tau = 0.1$

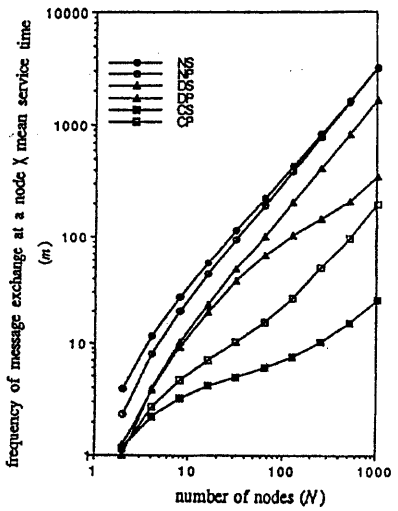

図3　ノード数の増加によるメッセージ数への影響

$\rho = 0.8, \tau = 0.1$

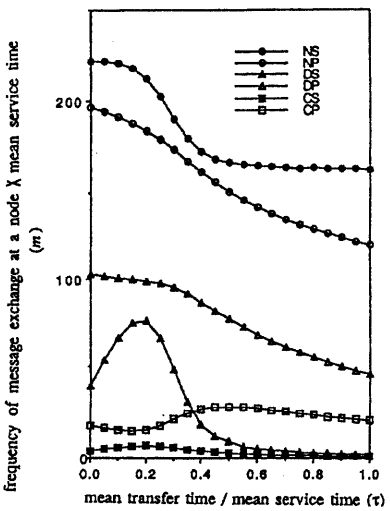

図4　ジョブ移送コストの変化の影響

$N = 64, \rho = 0.8$

謝辞

本稿の執筆にあたり、貴重な助言と協力を頂いた、前川守、清水謙多郎、亀田壽夫、ならびに前川研究室の諸氏に深く感謝する。

参考文献

[1] H. Ashihara, N. Mizuguchi and M. Maekawa: " All-Range Policies for Adaptive Load Sharingin Distributed Systems," in Proc. ISMM International Workshop on Parallel Computing, 1991, pp. 276-279.

[2] R. M. Bryant and R. A. Finkel: " A Stable Distributed Scheduling Algorithm," in Proc. 2nd International Conference on Distributed Computing Systems, 1981, pp. 314-323.

[3] D. L. Eager, E. D. Lazowska and J. Zahorjan: "Adaptive Load Sharing in Homogeneous Distributed Systems," IEEE Trans. Softw. Eng., Vol. SE-12, No. 5, 1986, pp. 662-675.

[4] D. L. Eager, E. D. Lazowska and J. Zahorjan: "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," Performance Evaluation, Vol. 6, 1986, pp. 53-68.

[5] K. Efe and B. Groselj: "Minimizing Control Overheads in Adaptive Load Sharing," in Proc. the 9th International Conference on Distributed Computing Systems, IEEE, 1989, pp. 307-315.

[6] A. Hac and W. Jin: "Dynamic Load Balancing in a Distributed System Using a Decentralized Algorithm," in Proc. 7th International Conference on Distributed Computing Systems, IEEE, 1987, pp. 170-177.

[7] K. J. Lee and D. Towsle: "A Comparison of Priority-Based Decentralized Load Balancing Policies," in Proc. Performance '86 and ACM SIGMETRICS, 1986, pp. 70-77.

[8] M. Livny and M. Melman: "Load Balancing in Homogeneous Broadcast Distributed Systems," in Proc. ACM Computer Network Performance Symposium, 1982, pp. 47-55.

[9] L. M. Ni, C-W. Xu and T. B. Gendreau: "A Distributed Drafting Algorithm for Load Balancing," IEEE Trans. Softw. Eng., Vol. SE-11, No. 10, 1985, pp. 1153-1161.

[10] J. A. Stankovic: " Simulation of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms, "Comput. Networks, Vol. 8, No. 3, 1984, pp. 199-217.

[11] Y. Wang and R. J. T. Morris: "Load Sharing in Distributed Systems, " IEEE Trans. Comput., Vol. C-34, No. 4, 1985, pp. 204-217.