C-019

市場メカニズムにもとづく QoS 適応リソース割り当てシステム QARAMAME の設計とシミュレーション

Design and Simulation of QARAMAME, a QoS-Adaptive Resource Allocation System Based on Market Mechanism

吉田 英樹† 崎山 伸夫†
Hideki Yoshida Nobuo Sakiyama

1. はじめに

複数の計算機を接続して処理を行なうクラスタシステムには、複数の計算機を使うことによって、高速に計算ができるほか、複数のアプリケーションプログラムを一つのクラスタ上で動かして CPU 時間などのリソースをやりくりすることで、アプリケーションごとにシステムを構築するよりハードウェアを削減できる利点がある。そのため、数値計算やサーバアプリケーションなどの従来の分野に加えて、リアルタイムアプリケーションの一種であるセンサ情報の信号処理分野でもクラスタの利用がはじまっている。とくに、ハードウェアの重量や電源容量の制限が厳しい組込リアルタイムシステムでは、クラスタによるハードウェア削減への期待が大きい。

しかし、従来のクラスタでリアルタイム性が要求される処理を行なう場合は、一時的な負荷上昇や障害を見込んでハードウェア量に大幅な余裕を設けるか、負荷上昇時や障害時に一部のアプリケーションを停止するかが必要で、限定されたハードウェア削減効果しか得られなかった。

われわれが研究を行なっているリアルタイムクラスタシステム QARAMAME (QoS-Adaptive Resource Allocation by MArket MEchanism) では、信号処理アプリケーションに特化して、状況に応じて解像度などのアプリケーションの QoS を制御することで、アプリケーションをできるだけ停止せずにハードウェア量を削減することを目指している。本稿では、QoS を考慮したリソース割り当てを、サプライチェーンモデルを用いた市場メカニズムによって分散して行なう方式と、市場取引のシミュレーションについて述べる。

2. 前提とするハードウェアおよびアプリケーション

今回の研究の前提とするリアルタイムクラスタシステムでは、ネットワークで接続された複数の計算機(ノード)で、複数のセンサからの入力データの処理を複数のアプリケーションで行なう。一つのアプリケーションのインスタンスが複数の計算機で動作し、CPU 負荷などに応じて処理が振り分けられる(図 1)。

各計算機では一つ以上の CPU が一つのリアルタイム OS のもとで動作している。リアルタイムクラスタは、これらの上で動作するミドルウェアレベルで実現される。

† 株式会社　東芝　研究開発センター

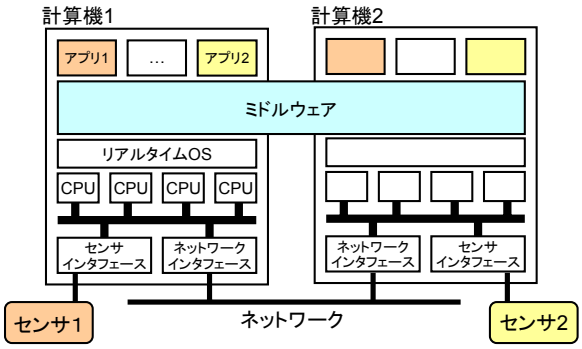

図 1　リアルタイムクラスタのハードウェア・ソフトウェア構成

3. QoS 適応による過負荷対応

クラスタでは、アプリケーションが大量の処理を行なったり、一部の計算機が故障したりすることによって、CPU の過負荷状態が生じることがある。リアルタイムシステムの場合、一定時間内に計算結果を出さなければならないため、過負荷への対応がとくに重要になる。

従来のクラスタでは、計算機間での処理の移動や負荷分散によって過負荷に対処していたが、これらの方法では負荷の総量は下がらないので、対応範囲は限られる。重要度の低いアプリケーションを停止することで負荷を下げる方法もあるが、アプリケーションを停止すれば、一部のサービスが使えなくなるといったユーザへの影響は免れない。

ここで、利用できるリソースの量に応じて、**サービスの品質**(quality of service, QoS)を上げ下げする **QoS 適応** [1]という手法がある。センサ信号処理では、画面のピクセル数や単位時間あたりのフレーム数といった値を QoS として扱い、負荷に応じて上下させることによってデータ量や処理量を増減させることができる。従来の QoS 適応は通信帯域や消費電力を増減させるために行なうことが多かったが、CPU 使用量を増減させることによって、クラスタの負荷を制御する効果も大きい。

この QoS 適応によって、故障で計算機数が減った、あるいは複数のアプリケーションのピーク負荷が重なったといった要因で、CPU 時間というリソースが不足した場合でも、QoS を低下させて処理を継続できる。逆にリソースに余裕があるときには、QoS を上げてリソースを活用することもできる。

従来 QoS 適応はアプリケーションが単独で行なうのが一般的だったが、クラスタミドルウェアと連動させれば、リソースの割当量を QoS 適応を考慮して決めることができ、リソースをより有効に活用することができる。

4. 市場メカニズムによるリソース割り当て

従来のクラスタシステムでは、クラスタ全体でリソースの集中管理を行ない、最適な割り当てを行なうのが一般的であった。しかし、クラスタ内の各ノードで多くのアプリケーションが動作する場合、これらのアプリケーションのリソース要求をすべて一箇所に集めて割り当てを行なうと処理のボトルネックが生じる恐れがある。さらに、現実のシステムでは、アプリケーションにリソースを割り当てるかどうかをアプリケーションごとに独立に決定するだけでは済まない場合が多い。たとえばあるアプリケーションが処理したデータを他のアプリケーションが利用する、といった依存関係がある場合、その依存関係によって生じるリソース割り当ての制約条件を反映した割り当てが必要となる。このような制約条件をすべて記述して集中管理するには、非常に複雑な割り当てシステムが必要となる。これらの制約条件も、できるだけローカルな処理で解決できることが望ましい。

また、クラスタでのリソース割り当ての計算は、最適解を求めようとすると 0-1 複数ナップサック問題[2]に対応し NP 困難となるため、なんらかの近似計算が必要である。

以上の課題に対応した、近似的なリソース割り当てをクラスタ上の計算機で分散して行なう方式として、本研究では市場メカニズムによるリソース割り当てを採用している。

市場メカニズムとは、アプリケーションやリソースマネージャなどのシステム構成要素がリソースをやりとりする際に、リソースの代価として仮想的な通貨を支払う機構である。リソースの値段と、どの構成要素がどの構成要素にリソースを渡すかは、構成要素間の市場取引によって決定される。構成要素がそれぞれ自分の利益(自分が供給するリソースの売値と自分が消費するリソースの買値との差)を最大化するように振舞うことで、システム全体のリソース割り当てが調整されるというのが市場メカニズムのポイントである。

このためクラスタ全体でリソースを集中管理する必要がなくなり、耐故障性や性能の面で利点がある。また、アプリケーション特有の制約条件などはリソースの入札方針として実装して入札内容に反映すればよく、システム全体への影響なしに各アプリケーションに閉じた処理で対応できるメリットもある。

5.サプライチェーン型市場メカニズム

リソース割り当てに市場メカニズムを導入する研究はすでに行なわれているが[3]、今回採用したサプライチェーン型市場メカニズムでは、CPU 時間などの物理的なリソース以外に、アプリケーションの計算結果として得られるデータなどもリソースとして扱い、値段をつけてやりとりする。たとえば、センサ信号の処理を行なうアプリケーションは、リソースマネージャから CPU 時間を買い、その CPU 時間を使ってセンサ情報を生成し、その情報を表示アプリケーションに売る(図 2)。

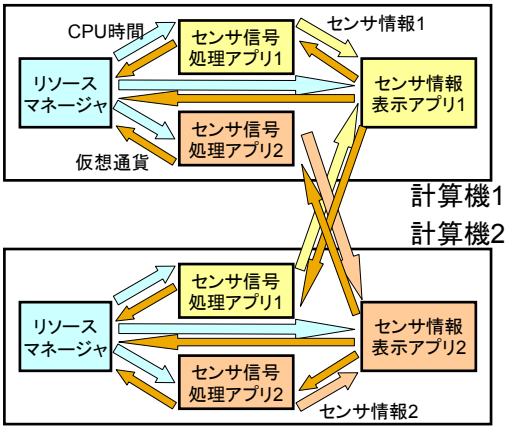

図 2　サプライチェーン型市場メカニズム

このため、最終的にユーザに提供されるリソースの値段を、ユーザにとっての重要度にもとづいてつけることで、ユーザにとって重要なデータを作るアプリケーションや、そのアプリケーションが利用するデータを作るアプリケーションに、CPU 時間などのリソースが優先的に割り当てられる。

また、リソースの値段をリソースの QoS によって上下させる(たとえば、解像度の高いデータであればより高い値段で買う)ことで、CPU 負荷とユーザにとっての重要度に応じた QoS 適応が自然に行なえる。

さらに、同じアプリケーションのインスタンスを複数のノードで動作させ、それぞれのインスタンスが計算結果のデータを売る入札を行なうことで、CPU 負荷が低くデータを低コストで生成できるノード上のインスタンスに処理を割り当てることができる。

サプライチェーン型市場メカニズムには以上のような利点があるが、一つのアプリケーションが複数の市場に対して入札を行なうため、それらの落札結果間で整合性が取れていないとアプリケーションが正しく動作できないという問題がある。たとえば、CPU 時間を買って処理を行なって処理結果のデータを売るアプリケーションの場合、処理結果データが売れたのに CPU 時間が買えないという落札結果になると、処理が行なえずに他のアプリケーションの動作にも影響が生じる。

このため、仮の入札を繰り返し、他の市場での落札結果を反映して入札内容を少しずつ調整しながら、システム全体として正しく動作ができる落札結果に収束させる、という方式を取る。収束したとみなすには、情報を買うアプリケーションが必要な情報をすべて買えているほか、CPU 時間を使用するアプリケーションが必要な CPU 時間をすべて買えている必要がある。

このような状態に短い時間で収束させるには、各アプリケーションによる入札価格や数量の調整方法、つまり入札方針がポイントになる。現在、CPU 時間を買って情報を生

成する信号処理アプリケーションのようなアプリケーションを「生産者」、表示アプリケーションのように複数の生産者から情報を買うアプリケーションを「消費者」と分類し、それぞれのカテゴリーごとに入札方針を作成している。

6. 市場メカニズムのシミュレーション

上記のサプライチェーンモデルの市場メカニズムのシミュレータを実装し、リアルタイムクラスタシステムのモデルを作成してシミュレーションを行なった。どのノードでどのアプリケーションが動き、それぞれどの市場への入札を行なうかをモデル記述として与えると、入札を繰り返して割り当て結果を出力するものである。

実際の運用システムにより近い多ノードでの多数のアプリケーションのシミュレーションも行なったが、ここではより小規模なモデルで QoS 適応を含んだシミュレーションを行なった結果について簡単に述べる。

このモデルでのアプリケーションとしては、2 つのセンサ(センサ 1・センサ 2)について、それぞれ信号処理アプリケーションと表示アプリケーションとがある。これらを各 2 つの CPU を持つ 2 つのノードで動作させる。

信号処理アプリケーションは 2 種類とも両方のノードにインスタンスを配置して負荷分散を行なう。一方、表示アプリケーションはノード 1 にセンサ 1、ノード 2 にセンサ 2 の表示アプリケーションを配置する。市場は各ノードの CPU 市場のほかに、センサ 1 情報市場とセンサ 2 情報市場とがある。

ここで、各アプリケーションの重要度を、各表示アプリケーションの予算の上限として与えて、表示アプリケーションが信号処理アプリケーションから買うセンサ情報の値段を制御し、それによって信号処理アプリケーションが買うことのできる CPU 時間を変化させている。

図 3 に示すシミュレーション結果では、センサ 1 の重要度(予算上限)を増やすとともにセンサ 2 の重要度を減らしてゆくと、センサ 1 の CPU 使用量が増えるとともにセンサ 2 の CPU 使用量が減っており、アプリケーションの重要度に応じた割り当てが行なわれていることがわかる。グラフを細かくみると、重要度の変化に対する CPU 時間の変化が単調ではなく一部で逆転が起きているが、これは、現在の入札戦略が収束性を重視したために重要度の細かい変化に追従できていないからだと考えられる。この点は今後の入札戦略の改善により解決可能と考えている。

割り当ては前述のように複数回入札を繰り返すことで収束させているが、その収束の過程を図 4 に示す。この例では、入札回数が 10 未満と非常に早く収束していることがわかる。他の例でも、入札は 15 回以内で収束している。

以上のように、重要度を反映したリソースの割り当てが少ない回数の繰り返し入札で行なえることをシミュレーションによって確認した。

7. おわりに

本稿では、市場メカニズムによってQoSを考慮したリソース割り当てを行なうリアルタイムクラスタについて、その概念とシミュレーション結果の説明を行なった。今後は、CORBA [4] (Common Object Request Broker Architecture)ミドルウェアの上での市場メカニズムを実現し、実際にアプリケーションが分散実行できるプロトタイプシステムを構築して評価を行なうほか、応用分野の拡大を図りたいと考えている。

参考文献

[1] Klara Nahrstedt, Hao-hua Chu and Srinivas Narayan, QoS-Aware Resource Management for Distributed Multimedia Applications, Journal of High Speed Networks, Volume 7, Issue 3-4, March 1999.

[2] Hans Kellerer, Ulrich Pferschy and David Pisinger, Knapsack Problems, Springer-Verlag, October 2004.

[3] Andrew Byde, Mathias Salle and Claudio Bartolini, Market-Based Resource Allocation for Utility Data Centers, HP Labs Technical Report HPL-2003-188, 2003.

[4] Steve Vinoski, Integrating Diverse Applications Within Distributed Heterogeneous Environments, IEEE Communications Magazine, Volume 14, Number 2, 1997.

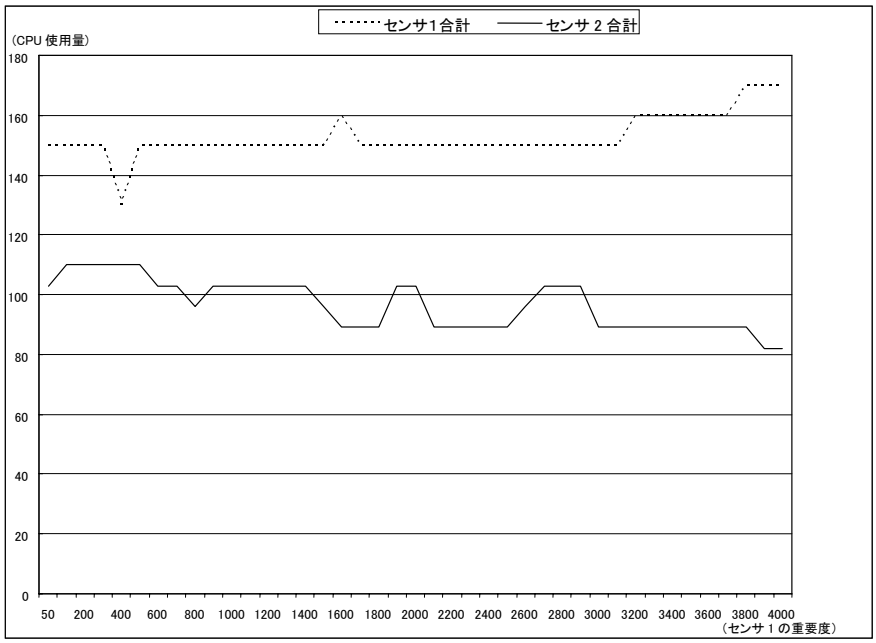

図 3　アプリケーションの重要度を反映した CPU 割り当て

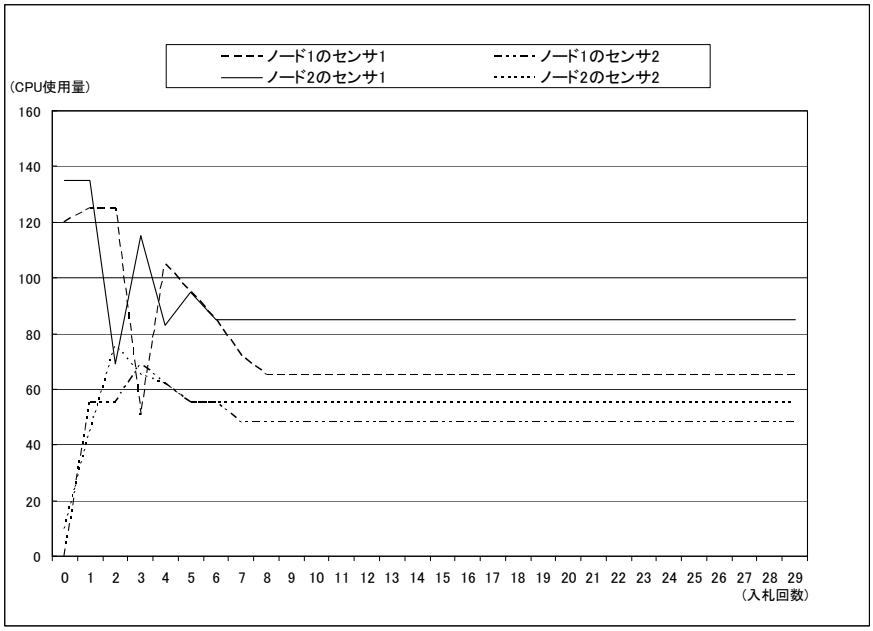

図 4　入札結果の収束