2007－GN－63 (17)
2007／3／22

双方向コミュニケーションサービスの品質を確保するリソース制御方式

大芝 崇　中島 一彰　金友 大　田淵 仁浩

NEC インターネットシステム研究所

概要

本稿では、遠隔ユーザ間で会話を行う双方向コミュニケーションにおいて、映像コーデックの変更や資料共有の同期制御方式の変更などの、アプリケーションの振舞いを変更する複数の処理に対して、アプリケーションレベルで実行順序制御を行うリソース制御方式を提案する。アプリケーションの振舞いを、映像中心の会話場面では映像を高精細化する一方で資料の同期精度を一段落とすように変更し、資料中心の会話場面では逆の振舞いをさせることができれば、伝えたい内容をより的確に伝えられる。しかし、これらの複数の処理を OS レベルの実行順序制御に任せて、安易に全端末で同時実行すると、低性能な端末や狭帯域ネットワーク環境下の端末において、CPU 資源や帯域などのリソースが不足することを排除できない。提案方式では、低性能・狭帯域な端末の余剰リソース量を考慮して、実行後にシステムの負荷が下がる処理を優先的に実行し、その結果増えた余剰リソース量を利用して高負荷な処理を実行することでリソース不足を防止する順序制御を行う。

Application-Level Task Scheduling based on Remaining Resource Amount for Real-time Communication Services

Takashi Oshiba, Kazuaki Nakajima, Dai Kanetomo and Masahiro Tabuchi

NEC Corporation, Internet Systems Research Laboratories

Abstract

We propose an application-level task scheduling for real-time communication services. In conventional approach, if multiple processes that change system behavior such as changing codec for video, changing method of drawing a sharing cursor for document sharing, etc., are executed simultaneously, resource shortages would occur in end systems with insufficient amount of CPU/network resources. In our approach, the multiple processes are scheduled in order to avoid resource shortages based on remaining resource amount in end systems with insufficient resources.

1. はじめに

近年、企業などの組織における意思決定の迅速化への期待から、音声通信、映像通信、資料共有などのコミュニケーション手段を統合するユニファイドコミュニケーション[1]が注目されている。意思決定の迅速化には、遠隔のメンバを一同に集めてリアルタイムに会話ができる双方向コミュニケーションが重要である。例えば、ユニファイドコミュニケーションにおける双方向コミュニケーションでは、IP 電話のような音声通信だけでなく、相手の表情を確認できる映像通信や、資料を見せながら説明箇所を共有ポインタで指し示せる資料共有や、相互に画面を操作して編集作業が行える画面共有などのアプリケーション(AP)を、伝えたい内容に応じて複数組み合わせて同時に利用できるため、意思決定の迅速化が期待できる。

単に複数の AP を同時利用するだけでなく、コーデックの変更による映像の高精細化や、画面転送方式の変更による画面共有の応答性能の向上などの、AP レベルでシステムの振舞いを変更[2]することで各 AP の品質を上下する

処理を、会話中に行うことができれば、伝えたい内容をより的確に伝えられる。例えば、製品の営業戦略会議において、製品を映して会話する場面では製品の映像を拡大して高精細に表示し、販売計画を説明する場面では図表を表示する画面共有の応答性能を向上する、ということができれば、意思決定のさらなる迅速化が期待できる。

しかし、このような複数のAPの振舞いを変更する処理（以下、APの振舞い変更処理と書く）を、OSレベルの実行順序制御に任せる形で安易に同時実行すると、低性能な端末や狭帯域ネットワーク環境下の端末で、処理の実行中にCPU資源や帯域などのリソースが不足する場合が生じる。その場合には、音切れやシステム全体の応答性能の悪化が発生し、ユーザの体感品質が劣化してしまう。

その理由は、例えば「製品の映像を拡大して高精細」にするためには、映像コーデックを高品質なものに変更する処理と、画面転送方式の変更により応答性能を一段低下させる処理とを行う必要があるが、これらの処理自体が比較的多くのCPU資源や帯域を必要とする一方で、通常のオフィス環境では、性能差のある端末と利用可能帯域の異なるネットワーク回線が混在するため、十分なリソースが利用できない端末が存在するからである。

そのため本稿では、低性能・狭帯域な端末の余剰リソース量を考慮して、APレベルで複数APの振舞い変更処理の実行タイミングに対する順序制御を行うリソース制御方式を提案する。

2. リソース制御における課題と要件

2.1. 従来技術の課題

APレベルでシステムの振舞いを変更する際のリソース制御に関する従来技術として、複数のAPを制御対象とするミドルウェアがCPU資源や帯域幅などのリソースを監視して、余剰リソース量が少なくなったらミドルウェアが各APに対して振舞い変更処理を要求する手法[2]がある。しかし、[2]では現在の余剰リソース量と、振舞い変更処理の実行後の余剰リソース量だけしか考慮されておらず、処理の実行中の余剰リソース量が考慮されていない。そのため[2]では、複数AP間で振舞い変更処理の実行タイミングを調整するような順序制御が行われず、実行中に多くのCPU資源や帯域幅を必要とする処理が、他の処理と同時に実行されてしまうことを排除できない。

ここで、複数APの振舞い変更処理の実行中のリソース不足を防止するために、この複数の処理を並列に同時実行するのではなく、完全に直列化して逐一実行する方法が考えられるが、この方法では処理数に比例して全処理の完了時間が増大し、全ユーザの待ち時間が長くなるため、リアルタイム性が重要視される双方向コミュニケーションには適さない。

2.2. リソース制御の要件

このような考察の結果、双方向コミュニケーションにおけるリソース制御では、複数APの振舞い変更処理を、品質劣化を起こさずに、かつ、ユーザの会話を中断させずに完了できる必要があると考えられる。そのため、次の要件を満たすことが重要である。

(1) **リソース不足の確実な防止**：処理実行後だけでなく処理実行中においても、低性能・狭帯域な端末におけるリソース不足を確実に防止できる。

(2) **全処理完了時間の最小化**：全てのAPの振舞い変更処理を短時間に完了させて、ユーザの待ち時間を最小化できる。

3. 提案するリソース制御方式

本稿では、低性能・狭帯域な端末の余剰リソース量と、各APの振舞い変更処理の実行により引き起こされる利用リソース量の変動量とをアプリケーション知識として利用した順序

制御を行うことにより、要件(1)(2)を満たすリソース制御方式を提案する。

3.1. 本方式の制御の概要

本方式で行う制御の目的は、CPU 資源の余剰量が最も少ない端末と、利用可能な帯域の余剰量が最も少ない端末の 2 台において、各振舞い変更処理の実行中と実行後の両方でリソース不足を確実に防止し、かつ、最小限の時間で処理を完遂することである。

そのために、まず、実行する複数の AP の振舞い変更処理が確定した時点における上記 2 台を特定する。次に、実行後にシステムの負荷が下がる振舞い変更処理を優先的に行い、その結果増えた余剰リソース量を利用して高負荷な処理を行う実行順序を決定する。複数パターンの実行順序が考えられる場合には、処理の実行時間の合計が最も少ないものを選択する。このような AP レベルでの順序制御により、先の 2 台の端末において、各振舞い変更処理の実行中と実行後の両方におけるリソース不足を防止しながら、最小限の時間で処理を完遂する。

3.2. データ構造

本方式で扱うデータ構造の全体像を図 1 に示す。図中の矢印はデータ間の依存関係を示す。

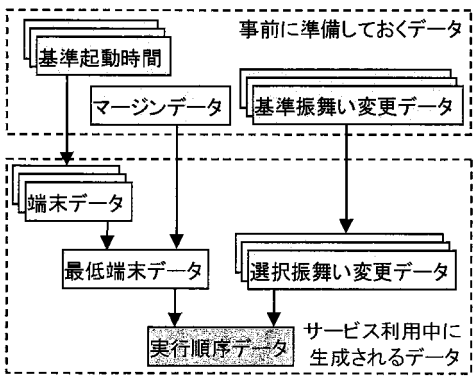

図 1：データ構造

前節で述べた制御を行うためには、各振舞い変更処理を実行した時に予測されるリソースの変化量と、各端末の処理性能やリソースの最新状態とを把握しておく必要がある。また、リソース不足を確実に防止するためには、常に余剰リソース量が枯渇しないようにマージンを持たせておく必要がある。

振舞い変更処理は複数個あるため、各振舞い変更処理を実行した時のリソースの変化量を端末毎に全て測定することは運用コストの面で非効率である。そのため本方式では、システムの導入時などに測定用の端末（基準端末と呼ぶことにする）を用いて各振舞い変更処理を実行した時のリソースの変化量を「基準振舞い変更データ」として事前に測定しておき、各端末は、この基準となるデータを元にして、各振舞い変更処理を実行した時に予測されるリソースの変化量を見積もる。

各端末の処理性能を比較するために、基準端末の処理性能を指標として用いる。各端末で AP の起動完了に要した時間と、事前に「基準起動時間」として測定済みの、基準端末での同じ AP の起動時間とを比較することで、端末間で処理性能の比較ができるようになる。

また、本方式ではシステムの運用者が「マージンデータ」の値を設定でき、余剰リソース量がこの値を下回ることがないように順序制御が行われる。

以下、それぞれのデータについて説明する。

3.2.1. 基準振舞い変更データ

システムに用意されている d 個の AP の集合を $D=\{1,\cdots,d\}$ とすると、基準振舞い変更データには、e 番目($e\in D$)の AP の振舞い変更処理を、単独で、基準端末で実行した時に、処理の実行中に観測された CPU 使用率と帯域幅の増分の最大値 smc_e と smb_e と、処理の実行の前後で上下した CPU 使用率と帯域幅の変動値 sac_e と sab_e と、処理に要した時間 st_e とが格納される。

3.2.2. 基準起動時間

e 番目の AP についての基準起動時間 ss_e には、基準端末において各 AP の単独の起動処理

にそれぞれ要した時間が格納される。ss_eは、端末毎の処理性能の指標を得るために利用される。

3.2.3. マージンデータ

マージンデータは、系内で1つだけ存在するデータであり、リソース不足か否かを判定する閾値として利用され、CPUマージンmg_cと帯域マージンmg_bとを格納している。

3.2.4. 端末データ

端末データは、端末毎に生成されるデータである。双方向コミュニケーションに参加中のh台の端末の集合を$H=\{1,\cdots,h\}$とすると、b番目($b\in H$)の端末の端末データは、端末の処理性能の指標であるidx_bと、現在のCPU使用率nc_bと、現在利用中の帯域幅nb_bと、現在の余剰帯域幅の予測値rb_bとを格納している。ここで、idx_bは、その端末がコミュニケーションを開始した際に起動したAPの起動時間をts_eとして$idx_b=ss_e/ts_e$で与えられる。この値により、端末間の処理性能の比較ができるようになる。またrb_bは、[3]の手法を用いて取得する。

3.2.5. 最低端末データ

最低端末データは、APの振舞い変更処理が実行される直前に生成されるデータであり、全端末のうち、その時点でidx_bの値が最小の端末のidx_bとnc_bと、rb_bの値が最小の端末のnb_bとrb_bとが、それぞれidx, nc, nb, rbとして格納される。

3.2.6. 選択振舞い変更データ

選択振舞い変更データは、APの振舞い変更処理が実行される直前に生成されるデータであり、実行することが確定した振舞い変更処理毎に生成される。実行を予定しているa個($a\le d$)の振舞い変更処理の集合を$A=\{1,\cdots,a\}$として、l番目($l\in A$)の選択振舞い変更データには、idx_bの値が最小の端末において予想されるCPU使用率の増分の最大値が$mc_l=smc_l/idx_b$として、CPU使用率の変動値が$ac_l=sac_l/idx_b$として、処理に要する時間$t_l=st_l/idx_b$として格納される。これらは、基準端末での測定値がidx_bの値によって補正されている。選択振舞い変更データには、これらに加えて、smb_lとsab_lの値が、それぞれmb_lとab_lとして格納される。

3.2.7. 実行順序データ

実行順序データには、各振舞い変更処理の実行順序が格納される。この実行順序は、例えば、振舞い変更処理を行うAPが5個ある場合に「最初に2番目のAPの振舞い変更処理を実行して、次に1番目と4番目を同時実行して、最後に3番目と5番目を同時実行する」順序である。これは、要素としてa個以下の振舞い変更処理を持ち、$V_1\cup V_2\cup\cdots\cup V_n=A$を満たす$n$個($n\le a$)の$A$の部分集合$V_i$を用いて、「$V_1$の要素の各振舞い変更処理を同時実行して、次に$V_2$の要素の各振舞い変更処理を同時実行して…」という処理をV_nまで続ける実行順序であると表現できる。先の例では$V_1=\{2\}$, $V_2=\{1,4\}$, $V_3=\{3,5\}$となる。

ここで、決定された実行順序で振舞い変更処理を実行中に、idx_bの値が最小の端末においてCPU資源は常に不足しないと仮定すると、V_iの要素のそれぞれはこの端末上では比較的低負荷な振舞い変更処理であるため、V_iの各要素を同時実行した時に要する時間を$time(V_i)$とすると、この仮定の下では、$time(V_i)$はV_iの各要素のt_lの最大値となる。

3.3. 実行順序の算出

前節で定められたデータを用いて、数式(2)～(5)の制約条件の下で、数式(1)で与えられる評価関数を最小化する、複数APの振舞い変更処理の実行順序を求める。

$$f(n)=\sum_{i=1}^{n} time(V_i) \tag{1}$$

$$\forall n \quad \sum_{i=1}^{n} vac(V_i)-vac(V_n)+vmc(V_n) \le 100-mg_c-nc \tag{2}$$

$$\forall n \quad \sum_{i=1}^{n} vac(V_i) \le 100 - mg_c - nc \tag{3}$$

$$\forall n \quad \sum_{i=1}^{n} vab(V_i) - vab(V_n) + vmb(V_n) \le rb - mg_b - nb \tag{4}$$

$$\forall n \quad \sum_{i=1}^{n} vab(V_i) \le rb - mg_b - nb \tag{5}$$

ただし、$vmc(V_i)$, $vac(V_i)$, $vmb(V_i)$, $vab(V_i)$ はそれぞれ、V_i の各要素の mc_l, ac_l, mb_l, ab_l の最大値を意味する。

この問題を次の手順で解く。ステップ1では、なるべく ac_l と ab_l が小さいものを優先して実行する順序を作ることで、前半で負荷を下げておき、後半で ac_l や ab_l が大きいものが実行できるようにする。ステップ2で、利用リソース超過を起こさないように並べ替えを行う。ここまでは並列実行をしない順序であるが、ステップ3で、並列実行と直列実行を組み合わせた順序について、数式(1)の値を調べる。以下、詳細な手順を述べる。

1. A の各要素を、点 (ab_l, ac_l) と直線 $y = -x - \beta$（β は十分大きな正の値とする）との距離 $g(l)$ の短い順に並べて、$g(l_1) \le g(l_2) \le \cdots \le g(l_a)$ とする。
2. for $l_k = 1$ to a do
   if（$l_k \ge 2$ & キューが空でない）then
   キューの先頭要素を r として取り出す
   if（$cac_{l_{k-1}} + mc_r \le 100 - mg_c - nc$ &
   $cac_{l_{k-1}} + ac_r \le 100 - mg_c - nc$ &
   $cab_{l_{k-1}} + mb_r \le rb - mg_b - nb$ &
   $cab_{l_{k-1}} + ab_r \le rb - mg_b - nb$）
   then 順列 σ の最後尾に r を追加
   else キューの先頭に r を戻す
   endif
   endif
   if（$cmc_{l_k} \le 100 - mg_c - nc$ &
   $cac_{l_k} \le 100 - mg_c - nc$ &
   $cmb_{l_k} \le rb - mg_b - nb$ &
   $cab_{l_k} \le rb - mg_b - nb$）then
   順列 σ の最後尾に l_k を追加
   if（$l_k == a$）then エラーで終了
   else キューの最後尾に l_k を追加
   endif
   endif
3. 全ての V_i の各要素を取り出した時に、各要素の順序が順列 σ の順序と一致する全ての V_i の組み合わせのうち、数式(1)が最小となるものを出力して終了。

ただし、cmc_{l_k} と cmb_{l_k} は、振舞い変更処理を1番目から順に $(l_k - 1)$ 番目まで実行してから、l_k 番目を実行した時のCPU使用率と帯域幅の増分の最大値であり、cac_{l_k} と cab_{l_k} は、振舞い変更処理の実行前と l_k 番目を実行した後でのCPU使用率と帯域幅の変動値である。

4. 評価と考察

本方式と、複数のAPの振舞い変更処理を一度に全て並列実行する方式（並列方式）と、振舞い変更処理を1つずつランダムに選んで直列実行する方式（直列方式）とを比較する。ここで、並列方式は、OSレベルで行う順序制御に相当する。

コミュニケーション中の複数の端末において、最低端末データが $nc = 80$ [%], $nb = 10$ [Mbps], $rb = 15$ [Mbps]で、マージンデータが $mg_c = 5$ [%], $mg_b = 5$ [Mbps]であるとする。表1に示す5つのAPの振舞い変更処理は、1章の営業戦略会議の例のような、AP間で利用リソース量を融通する場面を想定し、ac_l と ab_l の値に大小がある。これらの処理を実行した際のCPU使用率と帯域幅の推移を調べることで、APレベルで順序制御を行う本方式の有効性を評価する。

ここで、直列方式ではランダムな選択の結果、1, 3, 2, 5, 4の順番で実行している。また、本方式の実行順序は、この例では3.3節のステップ1で5, 3, 2, 1, 4、ステップ2で3, 5, 2, 1, 4、ステップ3で{3}, {5, 2}, {1, 4}と変化する。

表 1：評価に用いた5つの振舞い変更処理

l	1	2	3	4	5
mc_l [%]	20	6	10	24	25
ac_l [%]	+10	+5	-18	+20	-32
mb_l [Mbps]	3	4	6	6	2
ab_l [Mbps]	+2	-2	+1	+3	-5
t_l [sec]	0.8	0.9	0.5	1.2	1.5

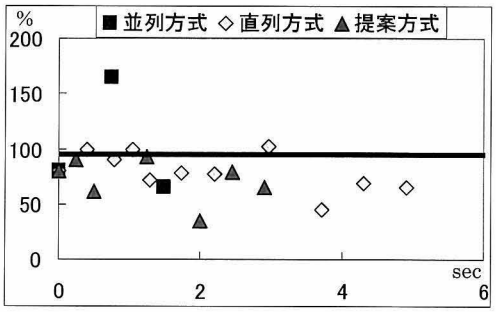

図 2：CPU 使用率の推移

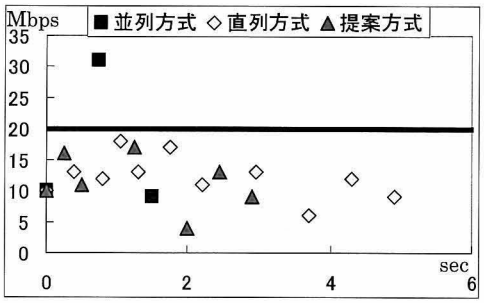

図 3：帯域幅の推移

並列方式、直列方式、本方式の cmc_{l_k} は 165, 102, 93[%]、cmb_{l_k} は 31, 18, 17[Mbps]、全処理完了時間は 1.5, 4.9, 2.9[sec]である。図 2 と図 3 の太線は $100 - mg_c = 95$ と $nb + rb - mg_b = 20$ を示しており、OS レベルの順序制御に相当する並列方式では CPU と帯域双方のリソース不足が発生している。また、直列方式では CPU リソースの不足が発生している。また、この全処理完了時間はリソース不足が発生しないという仮定の下での値であるため、両方式の実際の全処理完了時間はこれよりも長くなると考えられる。

本方式では、{3}と{5, 2}のような、ac_l と ab_l の小さい処理を優先して実行して前半（0～約 2 秒の区間）で利用リソース量を下げておき、その結果大きくなった余剰リソース量を後半で使うことで、{1, 4}のような ac_l や ab_l の大きい処理でも並列で実行できている。その結果、システム管理者がマージンデータにより設定した太線の値を超過することなく、リソース不足を防止しながら直列方式の 59%の所要時間で全処理を完了できている。

このように、AP レベルで順序制御を行う本方式の有効性が確認できた。

5. おわりに

本稿では低性能・狭帯域な端末の余剰リソース量を考慮して、AP レベルで複数の AP の振舞いを変更する処理の実行タイミングの順序制御を行うことで、ユーザの体感品質の劣化を防止するリソース制御方式を提案した。

今後の課題としては、実機での試験を通した実行順序の計算時間の測定と、双方向コミュニケーション以外の業務システムなどの AP も同時に利用する場合を対象としたリソース制御方式の確立などが挙げられる。

参考文献

[1] Gartner, Inc., "Magic Quadrant for Unified Communications, 2006", June 2006.

[2] Lakshman, K. et al., "Integrated CPU and Network-I/O QoS Management in an Endsystem", Proceedings of IFIP IWQoS '97, pp.167-178, May 1997.

[3] Eswaradass, A. et al., "Network Bandwidth Predictor (NBP): A System for Online Network Performance Forecasting", Proceedings of IEEE CCGRID '06, pp. 265-268, 2006.