Peer-to-Peer 環境における協調作業アプリケーションのための適応型調停プラットフォーム

吉永浩和† 土屋 健† 小柳惠一†

明示的なサーバが存在しない P2P ネットワークにおいて，複数の参加者で同期型の協調作業を行う場合，作業データに対する同期，一貫性，通信遅延の吸収，順序付けなどを保証しなければならない場合があり，参加者のいずれかの端末が一時的なサーバ（調停ノード）としてその役割を担当することで解決することができる．しかし，P2P ネットワークでは，ネットワークの不安定性，通信遅延，端末の相違などの特徴から，任意に調停ノードを決定することは難しい．本論文では，P2P ネットワークを構成する各ノードの環境情報やアプリケーションの目的に応じて，最適な調停ノードを，動的に決定する ACE（*Adaptive Coordinator Election*）プラットフォームを提案する．また，ACE は調停ノードに障害が発生した場合でも，アプリケーションへの影響を少なくするために，調停ノードを再配置することができる．筆者らは JXTA を用いて ACE を実装し，その動作確認および評価を行った．また，サンプルアプリケーションを実装し，その有効性を明らかにした．

Adaptive Mediation Platform for Collaborative Applications in Peer-to-Peer Environment

Hirokazu Yoshinaga,† Takeshi Tsuchiya† and Keiichi Koyanagi†

We propose the ACE (*Adaptive Coordinator Election*) platform that elects and relocates a coordinator adaptively in P2P networks. In collaborative applications, a coordinator mediates synchronization, consistency, sequencing and communication delay. However, it is difficult to decide a coordinator according to application's aim in P2P networks because of some features of network; unstable network, difference of physical networks and devices for participants. The ACE platform elects and relocates a coordinator dynamically according to environmental metrics obtained from participants. We implemented ACE platform and evaluated with JXTA framework. As a result, we confirmed a relocation of a coordinator adaptively in P2P networks.

1. はじめに

インターネット上で，Peer-to-Peer（以降 P2P と呼ぶ）ネットワークを利用した，Gnutella[10] や Freenet[11] などのファイル共有アプリケーションの人気が高まっている．これらはネットワークを構成するノードが分散してコンテンツを管理し，相互に通信することで，検索および取得することが可能になる．P2P ネットワークは，1. すべてのノードが同一の機能および責務を有している，2. すべての通信に対称性がある，という点で分散システムの 1 つの形態ととらえることができる．一方，1. ノードの出入りが頻繁に発生する，2. ノードが広域に分散している，3. ノード間で非同期に通信する，4. ノードやネットワーク環境の異なるヘテロジニアスな環境上にネットワークを構築する，という点で従来の分散システムとは異なる特徴を有している．本論文ではこれらの特徴を有する P2P ネットワークを P2P 環境と呼ぶ．

近年では，ファイル共有以外に，多くのアプリケーションが提案されている[12),13)]．Groove[12] は，非同期な作業を目的とした，多機能情報共有アプリケーションとして，企業などへの導入が進んでいる．また，skype[13] は，広域での音声通話を可能にしている．筆者らは Project JXTA[5] において，完全分散型の情報共有プラットフォームである JXCube（*Jxta eXstreme Cube*）[6] を提案している．JXCube[6] は，広域に分散したユーザ間で，グループベースの非同期・同期型の協調作業を可能にし，メッセージング，ストレージ，プレゼンス，セキュリティなどの機能を提供する．

† 早稲田大学大学院
Graduate School of Information, Production and Systems, Waseda University

協調作業時に相手が同時に存在し，リアルタイムで作業を行うチャットや会議システムなどの同期型協調作業では，データの同期，一貫性，通信遅延，順序付けなどの調停問題を解決する必要がある[14]．これらに対し，筆者らは，P2P 環境上で調停問題を解決するための動的な調停ノード決定アルゴリズム[1),2)] を提案している．また文献 1)，2) を用いて，音声や映像による同期型の協調作業を可能にする StarCast システム[3),4)] を提案している．

本論文では，P2P 環境において，同期型の協調作業を調停するために，動的にサーバの役割をするノード（以降，調停ノードと呼ぶ）を決定し，必要に応じて再配置する ACE（*Adaptive Coordinator Election*）プラットフォームを提案する．ACE は P2P 環境で調停ノードを決定するために，1. ヘテロジニアス環境を想定した環境情報による選択，2. 同一の作業者を論理的に集約するグループの作成，3. アプリケーションの利用目的に応じたポリシの定義，などが考慮されている．また，調停ノードの障害を動的に検出し，再配置することでシステムのフォールトトレランスを保証する．

本論文は以下の構成からなっている．2 章で関連研究の説明により，本論文で提案する ACE の位置付けを明確にし，3 章で ACE のアーキテクチャを示す．また，4 章でプラットフォームおよびサンプルアプリケーションの実装を示し，5 章で ACE の評価を示す．最後に，6 章で結論と今後の課題を示す．

2. 関 連 研 究

ACE は，動的に調停ノードを選択し，必要に応じて再配置する点において，従来の分散アルゴリズムを応用する．調停ノードの選任機構は，リーダ選択問題ととらえることができる．リーダ選択問題は，複数のノードの中から正確に 1 つのノードをリーダとして選ぶ問題であり，多くの研究がなされている[15),16)]．これらの選択プロトコルは，すべてのプロセスの中で，最大の ID を持つノードをリーダとする．また，静的なネットワークを対象としており，最終的にリーダが 1 つだけ選択されることを保証する．ACE でも，調停ノードを 1 つだけ選択する点において同様であるが，P2P 環境の特徴を考慮すると，そのまま適用することはできない．

また，アドホックネットワークを対象としたリーダ選択問題として文献 17)，18) がある．アドホックネットワークの場合，通信可能な範囲内のノード群でグループを形成し，そのグループの中からリーダを選択する．文献 17) は，ノードの移動を考慮した場合としない場合の 2 種類のアルゴリズムを提案している．前者は 1 つのノードがリーダになることを保証しないため，複数のノードがリーダになる可能性があり，後者は各ノードがランダムウォークすることを前提としているため，ノードの移動によるグループの分離や結合を考慮していない．また，文献 18) は，グループの分離や結合が発生しても，グループ内でリーダが 1 つ存在することを保証するが，その選択時間を考慮していないため，グループが変化した直後はリーダが存在しない可能性があり，その間にアプリケーションで発生した通信は考慮していない．また，文献 19) は，トポロジの変化に適応して動的に中央付近にいるノードをリーダに選択するアルゴリズムを提案している．しかし，トポロジの変化にともなうグループの分離や結合を考慮していないため，分断されたノードが継続して調停ノードを利用できることを保証していない．

オーバレイネットワークを形成する P2P ネットワークの場合，ノードの移動が発生した場合でもノード間の通信が保証され，グループの分離や結合を考慮する[5)]．これにより，ACE では，グループによる調停ノードの決定を行うことが可能である．

また，グループを利用したリーダ選択問題として文献 20) がある．文献 20) は，IP マルチキャストを利用して，パケットロスを考慮した *Probabilistic failure model* によってリーダを選択する．ACE では，グループメンバへのメッセージ配信に，高信頼なアプリケーションレベルマルチキャスト[8)] を利用するため，パケットレベルの喪失に関しては考慮する必要はない．また，文献 21) は，ノードのローカル情報に基づいてグループ内で投票を行うことで最適なリーダを選択するアルゴリズムを提案している．文献 21) は，最初にリーダが設定したパラメータに基づいて，全ノードの投票によってリーダを選択する．ACE では，同様にノードのローカル情報（環境情報）を用いるが，全ノードによる投票手法ではなく，調停ノードのみによる選択を行う．

システムを高信頼化するために，調停ノードのクラッシュやリンクの障害を検出する必要がある．これは，同期網を前提とした研究として，文献 22)〜25) などが提案されている．また，非同期網を前提とした研究として，文献 26)，27) などが提案されている．これらは通信のタイムアウトおよびノード間の合意によって障害を検出する．ACE では，文献 26) などと同様に，メッセージのタイムアウトによって障害を検出する．

3. アーキテクチャ

3.1 基本要件および前提条件

本論文で提案する調停ノード選任および再配置プラットフォーム（*ACE: Adaptive Coordinator Election*）の基本要件を以下に記す．

(1) ネットワークの特徴を考慮した調停ノードの決定
P2P 環境の特徴を考慮して調停ノードを決定するためには，ノードの環境情報（メトリック）に適応して動的に調停ノードを決定する必要がある．

(2) 調停ノードのクラッシュおよび通信の障害に対するフォールトトレランスの保証
現在利用している調停ノードがクラッシュした場合や調停ノードと参加ノード間の通信に障害が発生した場合，アプリケーションを停止することなく，調停ノードを再配置することによって，継続して利用できる必要がある．

(3) 協調作業アプリケーションに対して透過的な調停ノードの再配置
調停ノードが再配置される場合，ユーザにアプリケーションの再接続を要求することなく，継続して利用できる必要がある．これは再配置中でもアプリケーションデータの損失などの影響を少なくし，再配置後も継続してアプリケーションを利用できる必要がある．

次に，ACE の実現における前提条件を以下に記す．

(1) 参加ノードへのメッセージ通知に高信頼なアプリケーションマルチキャストを利用する．
(2) 参加ノードの障害を検出しない．
(3) 調停ノード選択に用いるポリシ定義は，あらかじめグループメンバで共有されている．
(4) 調停ノードに障害が発生した場合，バックアップ調停ノードに切替えを行うが，このとき，アプリケーションを継続するためにはチェックポイント[29] やメッセージログ[29] を各ノードが管理する．
(5) 最適調停ノードに移行する場合，アプリケーションを継続するために，Seamless Service[9] を利用する．

3.2 システム構成およびプロトコル

システムは，複数のノードから構成され，ノード間で連携することで，調停ノードの選任および再配置を行う．システム構成を**図 1** に示す．ノード間は ACE のコネクションを用いて P2P 通信をし，相互に接続される．そして，セッションによって調停ノードのアプリケーションと参加ノードのアプリケーションを接続し，調停機能を介したアプリケーションの利用が可能になる．また，ACE の構成要素は複数のモジュールからなる．MessageManager は，ACE のメインモジュールとして機能し，P2P 通信，グループ形成，障害検出，競合検出・解決，タイマなどを管理する．RelocationManager は，MessageManager と ElectionManager から再配置命令を受け，再配置を行う．ElectionManager は，MessageManager からの命令により，参加ノードからのメトリックを集計し，調停ノードおよびバックアップ調停ノードの選任を行う．MetricManager は，MessageManager からの命令により，ポリシファイルを参照し，メトリック値を取得する．MediationManager は，ACE を用いたアプリケーションのインタフェースを提供し，アプリケーションと ACE 間の連携機能を提供する．

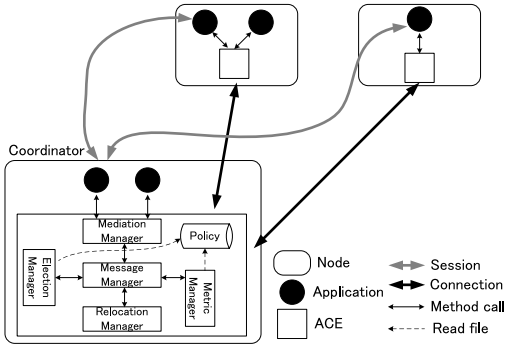


図 1 システム構成
Fig. 1 System architecture.

ACE が起動してから調停ノードを選任するまでのプロトコルを**図 2** に示す．ACE を起動後，グループに参加し，アプリケーションマルチキャストを用いて，現在の調停ノードを検索する．このとき，調停ノードが存在する場合には，検索結果からそれを取得する．また，存在しない場合には，自分自身が調停ノードになる．次に，各ノードはポリシファイルに定義されたメトリックを用いて，メトリック値を取得する．次に，各ノードは調停ノードから取得したメトリック送信時間（3.2.2 項）にメトリック値を送信する．次に，調停ノードは，参加ノードからメトリック値を受信し，確認メッセージを返信する．次に，調停ノードは，3.2.1 項の選任手法に基づき，調停ノードおよびバックアップ調停ノードを決定する．このとき，調停ノードは，一定時間経過後に受信したノード分のメトリック値を用いて，選任を行う．これは，調停ノードが参加ノー

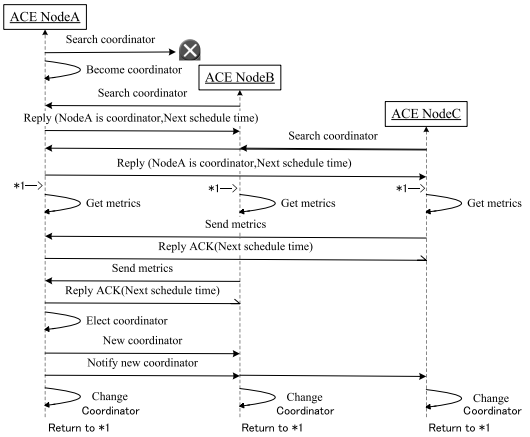


図 2 プロトコル
Fig. 2 Protocol.

ドの管理を行っていないためである．次に，選任結果の調停ノードが現在の調停ノードと異なる場合は，新規調停ノードに通知する．次に，新規調停ノードは，アプリケーションデータやセッションなどを前の調停ノードから取得し，参加ノードに調停ノードの変更を通知する．そして最後に，参加ノードはローカルに保存した調停ノード ID を変更する．

3.2.1 調停ノードおよびバックアップ調停ノードの決定（基本要件 1）

P2P 環境は，ネットワークを形成する端末の種類，ネットワーク帯域，通信プロトコルなどが異なるヘテロジニアス環境上に構築されるため，アプリケーションの利用目的に応じた調停ノードを任意に決定することができない．たとえば，多地点間遠隔会議システムでは，会話の整合性を保つために，ある程度の時間的な同期が必要であり，それを調停ノードに要求する．この場合，調停ノードのネットワーク帯域が大きく，レイテンシが低いものが最適であると考えられる．また，オークションやオンラインゲームでは，トランザクションなどの計算能力を必要とする処理が多く，計算処理能力のあるノードが調停ノードになることが望まれる．

ACE では，ヘテロジニアス環境に適応して最適な調停ノードを決定するために，参加ノードの環境情報（メトリック）を用いた選択を行う．メトリックは，1 つ以上選択することができ，1 つの場合，各参加ノードのメトリック値から最適値を持つノードを調停ノードに決定し，それ以外のノードはメトリック値の降順または昇順にバックアップ調停ノードにする．また，複数のメトリックを選択した場合は，参加ノードにおけるメトリックごとの順位を総計したものをコストとし，最小コストを持つノードを調停ノードに決定する．そして，それ以外のノードは，コストの昇順にバックアップ調停ノードにする．なお，コストが同一の場合には，最大の ID を持つノードを調停ノードにする．メトリックを利用した選択によって，全参加ノード N に対して，調停ノードが 1 つ，バックアップ調停ノードが $N-1$ つ存在することになる．次に，ACE で利用するメトリックを以下に記す．

レイテンシ レイテンシは，特定ノード間の RTT（*Round Trip Time*）が最も小さいノードを調停ノードにするメトリックである．これは，実ネットワークに負荷が高い場合に有効になる．また，実ネットワークの負荷が高い場合，調停メッセージや調停結果メッセージの通知に遅延が発生するため，調停ノードを再配置することによって，遅延を短縮することができる．

ネットワーク接続時間 ネットワーク接続時間は，統計的に最も長くネットワークに接続しているノードを調停ノードにするメトリックである．これは，P2P 環境のようなノードの出入りが頻繁に発生する場合に安定した接続を維持することが可能になる．

端末の処理能力 端末の処理能力は，処理能力の低いノードが調停ノードになることを避けるメトリックである．これは，PC，PDA，携帯電話など処理能力の異なる参加ノード間で，PDA や携帯電話などの処理能力の非力な端末に負荷がかかることを避けることができ，ヘテロジニアスな環境での利用に有効である．

制約条件 メトリック値による判断だけでなく，制約条件を利用して，調停ノードの決定および再配置が可能である．調停ノードの決定に制約条件を設定することで，複数のメトリックを用いた場合の優先度にすることができる．制約条件は上記のメトリックに加え，電池残量率と CPU 使用率を利用する．これらは，閾値を上下した場合にシステムが不安定になり，アプリケーションに影響を与えるものである．たとえば，電池残量率は，電源の確保が難しいモバイル端末の場合には，電池残量率の高いノードに再配置することで，アプリケーションを継続して利用できる．また CPU 使用率は，調停ノードの負荷が高い場合に，他のノードに再配置することによって，調停処理時間を削減することができる．ネットワーク接続時間と端末の処理能力以外のメトリックには，閾値に加え，メトリックの上昇率または下降率による判断が可能である．メトリック値の履歴を用いて単位時間あたりの変動率を算出し，定義値を上下した場合に再配置を行う．

3.2.2 メトリック送信間隔

参加ノードは，調停ノードに要求メッセージを送信

するときに，メッセージに次回の選任時間までの残り時間（*nexttime*）の要求を含めて送信する．そして，調停ノードからの応答メッセージには *nexttime* が含まれ，それを利用して，参加ノードはメトリックを送信する．各メッセージにその要求を付加することで余分なトラフィックを抑えることが可能になる．

3.3 高信頼な調停ノードの選任（基本要件 2）

ACE では，ネットワークに分散するノードが，ある特定のアプリケーションを利用するために，論理的なグループを動的に形成し，その中で調停ノードを選任する．グループが持つ主要な性質は，メッセージが，あるグループに送られた場合，そのグループ内のメンバがそのメッセージを受信することである．これにより，グループ内の 1 つのノードが故障しても，他のノードがそれに代わってその仕事を引き継ぐことができる[28]．

3.3.1 障害への対処

本項ではノードのクラッシュ検出および通信障害によるメッセージの喪失の対処について記す．なお，ノードのクラッシュ検出は調停ノードのみ考慮し，参加ノードは考慮しない．

調停ノードのクラッシュ検出 調停ノードがクラッシュした場合，アプリケーションが調停機能を利用できなくなるという致命的な障害になりかねず，それを瞬時に検出する必要がある．参加ノードは調停ノードから応答がないことを確認することによってクラッシュを検出する．ACE では，参加ノードが調停ノードに送信するメッセージごとにタイマを起動し，調停ノードからの応答メッセージを監視する．そして一定時間 W_2（詳細は 3.3.2 項で説明）経過しても応答がない場合，N 回の再送を行い，それでも応答メッセージがない場合に，調停ノードがクラッシュしたと判断する．これにより，クラッシュ検出時間（T_d）は，式 (1) となる．

$$T_d = nexttime + syncdelay + (N \times W_2) \quad (1)$$

ここで，調停ノードがクラッシュした時刻 T_c から次の調停ノードを選任するまでの残り時間を *nexttime*（$nexttime = T_e - T_c$，T_e は調停ノード選任時刻とする），通信遅延を *syncdelay* とする．この検出時間は，要求メッセージ以外に，定期的なヘルスチェックメッセージを利用することでさらに短縮させることも可能である．

要求メッセージの喪失 要求メッセージは，参加ノードから調停ノードに対してのメッセージと判断することができる．ACE ではメッセージの再送によって，要求メッセージの喪失を解決する．参加ノードが要求メッセージを送信したときにタイマを起動し，応答メッセージが返信されず，一定時間を経過した場合に，メッセージの再送を行う．このとき，メッセージにはシーケンス番号を付加し，そのメッセージが 1 回目の要求なのか再送なのかを判断する．

応答メッセージの喪失 応答メッセージの喪失は，要求メッセージのタイマ管理によって再送することで解決できるが，問題は参加ノードにとって調停ノードからの応答がない理由が判断できないことである．要求メッセージあるいは応答メッセージが失われたのか，単に調停ノードの応答が遅いだけなのか，これによって違いが生じてくる．しかし，ACE における要求メッセージは，繰返し等価と判断することができ，複数回の要求は調停ノードにとって問題はない．また，たとえ繰返し等価でない要求が今後発生したとしても，各メッセージにはシーケンス番号が付与されているため，調停ノードでは 1 回目の要求なのか，再送された要求なのかを判断でき，2 回目以降の要求メッセージを棄却するだけでよい．

3.3.2 メッセージタイマの管理

参加ノードがネットワークの広域に分散している場合，調停ノードのメッセージ受信に，通信遅延が発生する．また，オーバレイネットワークでは，ネットワーク環境の相違によるメッセージ中継やプロトコル変換などで通信遅延が発生する．これにより，調停ノードは，遅延が大きいノードからはデータが届かないと判断し，そのノードが協調作業に加わることができない可能性がある．ACE では，調停ノードのメッセージ待機時間（W_1）および各ノードの調停ノードから確認および応答メッセージの待機時間（W_2）は，式 (2) により設定する．

$$\begin{aligned} W_1 &= W_2 \\ &= \max\{RTT_i\} + Ace.WaitingTime \quad (2) \end{aligned}$$

ここで，$\max\{RTT_i\}(i : node\ number)$ は，全ノードの最大 RTT 値（最大通信遅延ノードのレスポンスタイム）とする．また，*Ace.WaitingTime* は，あらかじめ指定した固定時間とする．

3.3.3 競合解決

ACE では以下の要因により，同一グループ内で調停ノードが 2 つ以上存在する可能性がある．本論文ではこれを調停ノードの競合と呼び，本項ではその解決手法を記す．

(1) 参加ノードがメトリックを送信後，確認メッセージが届かず（タイムアウトになる），調停ノードがクラッシュしたと判断．

(2) ACE 起動後，調停ノードを検索したとき，検

索・応答メッセージが届かず，調停ノードが存在しないと判断．

(3) 複数の参加者が同時に協調作業に参加した．

(4) 調停ノードが変更されたときの調停ノードからの応答メッセージが喪失した．

(1)，(2)，(3) は物理的に調停ノードが 2 つ以上存在する場合であり，(4) は論理的に調停ノードが 2 つ以上存在する場合である．(1) の場合，バックアップ調停ノードが調停ノードに生存確認を行うことで競合を解決する．また，(2) および (3) の場合，1 つの参加ノードのみが独立している場合と完全に独立した別グループを形成する場合がある．前者は，いずれ属していないグループ（プライマリグループ）の調停ノードからメッセージが届いた時点で競合を検出し，プライマリグループの調停ノードを新規調停ノードとする．また，後者の場合，別グループの参加ノードが，プライマリグループの調停ノードからメッセージを受信したときに競合を検出する．この場合，それぞれの調停ノードの最大 ID を持つ方を優先する．そして，(4) の場合，参加ノードが以前の調停ノード（$C1$）に要求メッセージを送信し，$C1$ は現在の調停ノード（$C2$）を返答することで解決する．

3.4 再配置（基本要件 3）

調停ノードの再配置は，調停ノードのクラッシュ時および現在の調停ノードよりも最適なノードを発見後の移行時に発生し，本節ではこれらにおける調停ノードの再配置手法について議論する．

3.4.1 調停ノードのクラッシュ時

調停ノードの再配置は以下の手順で行われる．

(1) 参加ノードが調停ノードのクラッシュを検出．

(2) 全参加ノードへ調停ノードのクラッシュを通知．

(3) バックアップノード（2 番目）に新規調停ノードを依頼．

(4) バックアップノードは新規調停ノードを全参加ノードに通知．

ACE では，アプリケーションに透過的に調停ノードの切替えを行う．つまり，再配置によってアプリケーションが再接続を要求されることなく，継続して利用することができる．これは，**図 3** の手順で，メッセージキューを用いて，アプリケーションが調停ノードに送信しているメッセージを一時的に蓄積することで実現する．ACE では，アプリケーションからのメッセージを取得し，キューに蓄積（push）する．そして，調停ノードとのリンクが正常な場合にはキューからメッセージを取得（pop）し，送受信を行う．また，調停ノードの変更通知を受けると，新規調停ノードにリンクを変更する．メッセージ送信に失敗した場合には，メッセージは再びキューに追加され，調停ノードとのリンクが正常になったときに再送される．

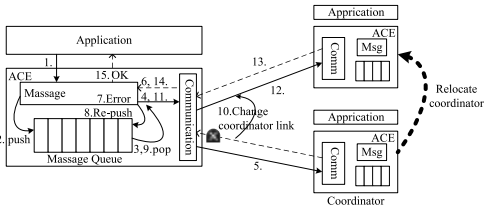


図 3 メッセージ送受信構造
Fig. 3 Messaging architecture.

3.4.2 現在の調停ノードよりも最適なノードを発見後の移行時

ノードの環境情報はつねに変化するため，定期的な調停ノードの選任結果が，それぞれ異なる可能性がある．しかし，そのつど調停ノードを再配置した場合にはオーバヘッドが発生し，それが逆にアプリケーションの動作を不安定にする可能性がある．ACE では，再配置するために再配置条件として環境情報の閾値を設定することでこれを解決する．現在の調停ノードが実行中は，環境情報の閾値を超えない限り継続して利用することが好ましい．しかし，閾値を上下し，調停ノードを再配置する必要がある場合には，クラッシュ時と同様にアプリケーションに透過的な再配置を実現する．

3.5 システムの限界

ACE では以下の点を達成することができず，システムの限界として扱い，今後の課題とする．

(1) グループ内で調停ノードの競合が発生し，その間に調停作業が行われる場合，競合が解決されるまでの間 total ordering などの調停を保証できない．

(2) グループの参加ノードは，他のノードを管理していないため，全参加ノードによる合意手法を用いた正確な調停ノードの障害検出ができない．

なお，(1) は，競合中は total ordering などの調停を保証できないが，競合が解決された時点で，調停を保証することが可能である．また，(2) は，グループが分離し，競合が発生する可能性があるが，これは，3.3.3 項で説明した競合解決手法を用いることができる．

4. 実　　装

4.1 ACE サービス

ACE は協調作業アプリケーションと独立したミド

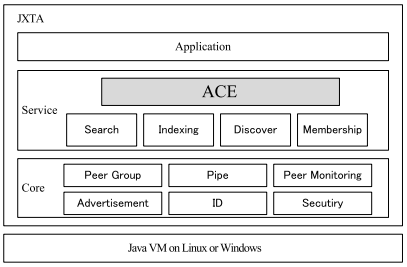


図 4 ソフトウェア構造
Fig. 4 Software architecture.

```
ACE root
 | - bin - ACE module
 | - lib
 | - app - app1
            | - policy file
            | - binary
       - app2
            | - policy file
            | - binary
       - app3
            | - policy file
            | - binary
```

図 5 ディレクトリ構成
Fig. 5 Directory constitution.

ルウェアとして実装する．**図 4** にソフトウェアアーキテクチャを示す．ACE は，JXTA アーキテクチャの Service レイヤで，JXTA サービスとして実装され，Core レイヤの通信（Pipe）やグループの形成（Peer group），また，Service レイヤの検索（Search）や発見（Discover）などの機能を用いる．そして，Application レイヤでは ACE を用いた協調作業アプリケーションが実装される．A 実装には Java 言語を採用し，*JRE*1.4.2 および *JXTA J2SE*1.3.1 で動作確認を行った．また，**図 5** は ACE のディレクトリ構成である．*ACE root* を基点として，*bin* ディレクトリに ACE サービスの jar ファイルを，*lib* ディレクトリに JXTA で必要とする jar ファイルを，*app* ディレクトリに，各アプリケーションをサブディレクトリに分け配置する．

ACE では，P2P ネットワークの構築に，オープンソースプロジェクトの JXTA を用いた．JXTA は，PC，PDA，携帯電話など様々な端末で動作し，オーバレイネットワークを形成することで，NAT や Firewall などを意識せず，端末間で相互通信が可能になる．JXTA の構成要素は，ピア，ピアグループ，サービス，通信，発見に分類される．ピアは JXTA ネットワークを構成するノードを指し，その処理能力や実ネットワークの形態により，ランデブーピア（リソースの発見），リレーピア（ルーチング）そしてエッジピアに分類される．ACE ノードは端末ごとに JXTA によって自動的にこれらに分類される．ピアグループは，ピアの論理的な集合を指し，ピアグループ内でのイベントは参加ピア間で共有される．ACE は，利用アプリケーションごとにピアグループを作成し，その中で調停ノードが決定される．サービスは，ピアおよびピアグループが提供する機能群を指し，ACE はピアグループが提供するサービスとして実装する．通信はパイプと呼ばれる仮想的な伝送路をピア間で確立することでメッセージの送受信が可能になる．送信用の OutputPipe，受信用の InputPipe を利用し，1 : 1 通信のための UnicastPipe，1 : N 通信のための PropagatePipe，双方向，同期型 Bi-DirectionalPipe がある．ACE は調停ノード間通信に UnicastPipe を，グループ内通信に PropagatePipe を，アプリケーションに調停ノード間の通信路を提供するために Bi-DirectionalPipe を用いる．発見は JXTA で定義されるリソースの検索・発見を指し，SRDI アルゴリズム[8] によって，スケーラブルな発見が可能になる．ACE ではピア，ピアグループそしてパイプの発見に利用する．

```
# ACE Properties
Ace.Version = 1.0.0
Ace.Enabled = yes
Ace.WaitForSearchTime = 10000
Ace.SearchMessageNum = 1
Ace.WorkerPeriod = 60000
Ace.ConstTimeForWaiting = 3000
Ace.RetransmitMessageNum = 0

# Metric
Metric.Num = 1

Metric.1 = metrics.NetworkConnectionTime
Metric.2 = metrics.latency
Metric.3 = metrics.DevicePower
Metric.4 = metrics.CPUUsage
Metric.5 = metrics.BatteryPower

# Restriction
Metric.5.Threshold <= 30
Metric.4.Threshold >= 80
Metric.4.Regulation >= 75
RegulationTime = 10000
```

図 6 ポリシ定義
Fig. 6 Policy definition.

4.1.1 ポリシの定義

ポリシはアプリケーションが調停ノードの選任で用いる利用メトリック，制約条件などを定義したもので，ACE では開発者がアプリケーションごとにポリシファイル（*policy.properties*）を用意する．**図 6** にポリシファイルの記述例を示す．

ポリシファイルの定義は大きく分けて，ACE 全般とメトリック全般の設定に分類される．ACE 全般の設定として，*Ace.WaitForSearchTime* は初期起動時にグループ参加ノードの中から現在の調停ノードを検索するときの最大待ち時間で，ミリ秒単位で設定する．*Ace.SearchMessageNum* は調停ノードの検索メッセージの再送回数を設定する．*Ace.WorkerPeriod*

は調停ノードの選任間隔で，ミリ秒単位で設定する．*Ace.WaitingTime* はメッセージの応答待ち時間の固定値をミリ秒単位で設定する．この値に各ノード間の RTT 値の最大時間（$\max\{RTT_i\}$）を加えた値を利用する．*Ace.RetransmitMessageNum* はメッセージの再送回数を設定する．

メトリック全般の設定として，*Metric.Num* は利用するメトリックの数を設定する．*Metric.N*（N はメトリック番号）は利用するメトリックの実装クラス名を設定する．*Metric.N.Threshold* は指定したメトリックの制約条件の閾値を設定する．*Metric.N.Regulation* は単位時間あたりの変動率を設定する．*RegulationTime* は変動率を求める単位時間を設定する．たとえば，図 6 の制約条件は，電池残量率が 30%以下，CPU 使用率が 80%以上または 10 秒あたりの上昇率が 75%以上になった場合を意味する．

4.1.2 メトリック

ACE では，実装の効率化とメトリックの独立性を高めることを目的として，それぞれのメトリックの実装に Java 言語を利用する．すべてのメトリックは *AceMetric* インタフェースを実装し，各ノードでは一定間隔ごとに *getMetric()* を実行し，ポリシファイルに定義されたメトリック値を取得する．その後，すべてのメトリックをラップした *AceMetrics* オブジェクトとして調停ノードに送信する．調停ノードは受信したメッセージからそれぞれの *AceMetric* オブジェクトを取得し，オブジェクトの持つ *ranking()* によって，全ノード内での順位付けを行う．以下に各メトリック値の取得方法を記す．

レイテンシ レイテンシの計測には RTT を利用する．32 byte のデータを定期的に特定のノードへ送信し，そのノードからの応答時間の平均値を利用する．

ネットワーク接続時間 ACE を起動してからの経過時間を利用する．ACE を長時間起動しているノードが最も長くネットワークに接続しているノードと判断する．

端末処理能力 処理能力は，起動時にベンチマークプログラムを実行させた実行時間を用いる．ベンチマークプログラムは Java 言語で実装し，10,000 個の *String* データからなる配列を *for* ループによって別の配列にコピーするのに要した時間とした．ただし，Java での時間計測は *System.currentTimeMillis* を用いるが，各 OS でクロックの精度が異なるため，100 msec の精度で実行時間を正規化した．

電池残量率，CPU 使用率 電池残量率と CPU 使用率の取得は，OS に依存するため，Linux と Windows 用の実装を行った．Linux 実装では，proc ファイルシステムを利用し，Windows 実装では WindowsAPI を利用した．

```
public abstract class Peerlet implements Runnable {

    protected Peerlet() {
    }

    // クライアントアプリケーションからのメッセージ処理するために、
    // Peerletクラスを継承したクラスで必要なビジネスロジックを実装
    public abstract void doInput(String peerID          //送信クライアントID
                                 , String peerName      // 送信クライアント名
                                 , AceMessage msg       // メッセージ
                                 , long time            // 受信時間
                                 , JxtaBiDiPipe pipe    // パイプ
                                 );

    public void start(AceService ace) {
        // ミドルウェア上にPeerletクラスを継承した独自クラスを配置すると
        // 自動的に呼び出され、JxtaServerPipeを作成
        // その後、スレッドとして実行
        Thread thread = new Thread(this);
        thread.start();
    }

    public void run() {
        // クライアントからのメッセージを待機
        // accept後、receiveMessageメソッドの呼び出し
    }

    private void receiveMessage(JxtaBiDiPipe pipe) {
        // クライアントアプリケーションからのメッセージを受信
        // その後、doInputメソッドからの呼び出し
    }
}
```

(a) Peerlet クラス

```
public abstract class PeerClient implements PipeMsgListener, Runnable {

    protected PeerClient() {
        // スレッドとして開始
        Thread aceServiceThread = new Thread(this);
        aceServiceThread.start();
    }

    public void run() {
        // 調停ノードのパイプアドバタイズメントの読み込み
    }

    public void sendMessage(AceMessage aceMsg) {
        // 調停ノードへメッセージを送信
    }

    public void pipeMsgEvent(PipeMsgEvent event) {
        // 調停ノードからメッセージを受信
            その後、messageRecievedメソッドの呼び出し
    }

    public void addPeerletMessageListener(PeerletMessageListener listener) {
        // クライアントアプリケーションが調停ノードからのメッセージを
        // 取得するためにPeerletMessageListenerオブジェクトを登録
    }

    public void removePeerletMessageListener(PeerletMessageListener listener) {
        // PeerletMessageListenerオブジェクトの登録を解除
    }

    protected void messageRecieved(AceMessage evt) {
        // PeerletMessageListenerを登録したオブジェクトに対して
        // 調停ノードから受信したメッセージを通知
    }

}
```

(b) PeerClient クラス

図 7 API
Fig. 7 API.

4.1.3 API

開発者は，ACE が提供する API を利用して，調停ノードを用いたアプリケーションを作成することができる．その実装には，調停ノードが実行する *Peerlet* クラスと参加ノードが実行する *PeerClient* クラスを用いる．**図 7** に *Peerlet* クラスと *PeerClient* クラスを示す．

***Peerlet* クラス** *Peerlet* クラスは，調停ノードで実行するための抽象クラスを提供する．調停機能の利用目的は各アプリケーションによって異なるため，それらの機能は提供せず，開発者が任意に実装する．*Peerlet* クラスは調停ノード上で実行され，*PeerClient* クラス間の双方向，同期型通信を確立する．また，*PeerClient* クラスから受信したメッセージを，アプリケーションクラスへリダイレクトするために，抽象型の *doInput()*

```
public class BBSService extends Peerlet {

    // 掲示板スレッド保存用ハッシュテーブル
    private Hashtable threadTable;

    public void doInput(String peerID, String peerName
            , AceMessage msg, long time, JxtaBiDiPipe pipe) {

      // 掲示板メッセージオブジェクト取得
      BBSMessage bbsmsg = (BBSMessage) msg;

        // スレッドIDとメッセージを取得
        BigInteger threadID = bbsmsg.getThreadID();
        String message = bbsmsg.getMessage();

        // スレッド単位、時間順にメッセージをハッシュテーブルに保存
        setMessage(threadID, time, peerName, message);

        // 調停ノードの管理するスレッドを含めて応答メッセージを作成
        Message resmsg = createMessage(threadTable);

        // 応答メッセージの送信
        pipe.sendMessage(resmsg);
    }
}
```

(a) BBSService クラス

```
public class BBSClient extends PeerClient implements PeerletMessageListener {

    // クライアントプログラム開始
    public static void main(String args[]) {
        BBSClient client = new BBSClient();
    }

    public BBSClient() {
        // PeerletMessageListenerの登録
        this.addPeerletMessageListener(this);
        initComponents();
    }

    private void initComponents() {
        // GUIコンポーネントの作成
    }

    // PeerletMessageListenerインタフェースのmessageRecievedメソッドを実装
    // 調停ノードからのメッセージを受信し、処理を行う
    public void messageRecieved(AceMessage msg) {
        // 掲示板用メッセージオブジェクト取得
        BBSMessage bbsmsg = (BBSMessage) msg;

        // 調停ノードから全メッセージを解析し、GUIに表示
    }
}
```

(b) BBSClient クラス

図 8 掲示板システムの実装例
Fig. 8 Example of BBS system.

を実装する．

***PeerClient* クラス** *PeerClient* クラスは，参加ノードで実行するための抽象クラスを提供する．*PeerClient* クラスは，参加ノード上で実行され，*send()* を用いて，調停ノードにメッセージを送信する．また，調停ノードから受信した返信メッセージを，アプリケーションクラスにリダイレクトするために，*addPeerletMessageListener()* によってメッセージリスナを登録し，受信したメッセージを処理する *messageRecieved()* を実装する．

4.2 アプリケーションの実装例

ACE を用いたシステムが構築可能であることを示すために，掲示板（BBS）システムを実装した．従来の掲示板は，WEB サーバがスレッドの順序一貫性や因果一貫性を保証する．本システムは，ACE を利用することで，調停ノードがその役割を担当する．*Peerlet* クラスを用いた *BBSService* クラスと *PeerClient* クラスを用いた *BBSClient* クラスの実装例を**図 8** に示す．スレッド順序は参加ノードから受信したタイムスタンプによって管理し，因果関係は，メッセージのスレッド ID を用いて実現した．

5. 評　　価

ACE を用いて実験，評価を行った．利用した実験環境を**図 9** に示す．なお，各実験において利用した端末構成は以下のとおりである．選任時間と障害回復の評価は，ノード数別に，2:A-E，3:A-D-E，4:A-B-D-E，5:A-B-C-D-E，6:A-B-C-D-E-F（2 から 6 はノード数：A から F は図 9 のノードに対応）を利用し，オーバヘッドと適応性の評価は 3:A-D-E を利用し，スケーラビリティの評価は，A-B-C-D（C が調停ノード）を利用した．

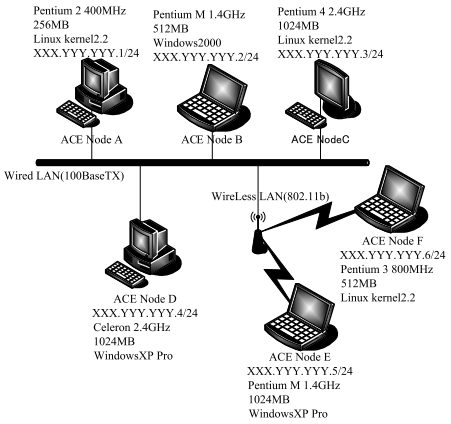
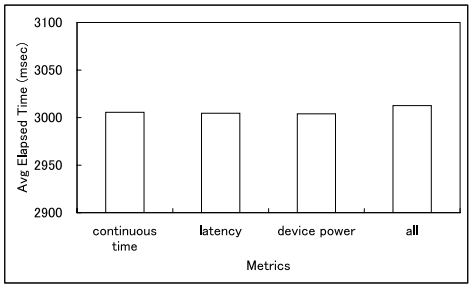


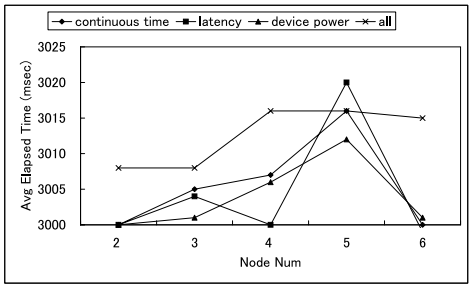
図 9 実験環境
Fig. 9 Experimental setup.

5.1 選 任 時 間

調停ノードが選任されるまでの時間を計測した．計測には，現在の調停ノードが最初のノードからメトリック値を受信してから，調停ノードを選任し終えるまでの時間を計測した．また，ノード数を 2 から 6 まで変化させ，メトリックをネットワーク接続時間，レイテンシ，端末処理能力，すべてを利用した場合を用いて，10 回の選任時間の平均を計測した．パラメータは *Ace.WorkerPeriod* を 60,000 msec に，*Ace.WaitingTime* を 3,000 msec に，*Ace.RetransmitMessageNum* を 0 回に設定した．メトリック別の選任時間を**図 10** (a) に示す．各メトリックの平均は，それぞれ，3,002 msec，3,001 msec，3,004 msec，3,013 msec であり，3,000 msec 程度の選任時間がかかっていることが分かる．しかし，それぞれの 3,000 msec は *Ace.WaitingTime* の 3 sec を含むため，選任に必要な時間はミリオーダで解決し，メトリック間での違いがないことが分かる．また，選任時間は *Ace.WaitingTime* に依存するため，大きい値を設定すると参加ノードに通知されるまでに時間がかかる．また，パラメータは *Ace.WorkerPeriod* > *Ace.WaitingTime* に設定する必要がある．また，図 10 (b) はノード数を 2 から 6 まで増加させた場

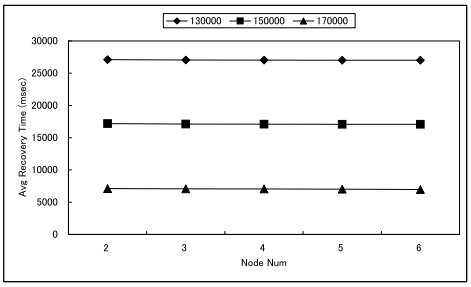


(a) メトリック別

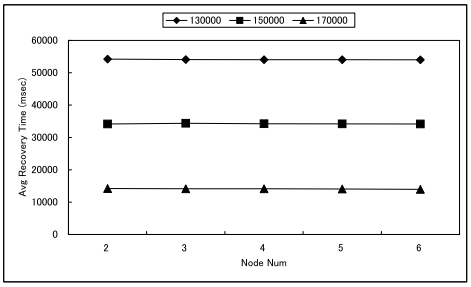


(b) ノード数別選任時間

図 10 選任時間
Fig. 10 Election time.



(a) 3,000 msec



(b) 6,000 msec

図 11 障害回復時間
Fig. 11 Recovery time from failure.

合の平均選任時間を示している．平均 3 sec かかり，6 ノード程度であれば，ノード数に依存していないことが分かる．

5.2 障害からの回復

調停ノードに障害が発生した場合に，再配置することを確認するために障害からの回復時間を測定した．また，ノード数を 2 から 6 まで変化させ，メトリックにネットワーク接続時間を用いて，10 回の平均を計測した．また，パラメータは *Ace.WorkerPeriod* を 60,000 msec に，*Ace.WaitingTime* を 3,000 msec および 6,000 msec に，*Ace.RetransmitMessageNum* を 0 回に設定し，調停ノードをクラッシュさせるタイミングを実験開始，13,000 msec，15,000 msec，17,000 msec（それぞれ，2 回目の選任時間の 10 sec 後，2 回目の選任時間の 30 sec 後，2 回目の選任時間の 50 sec 後）とした．**図 11** (a) は *Ace.WaitingTime* が 3,000 msec のとき，図 11 (b) は 6,000 msec のときのクラッシュ障害からの回復時間を示している．*Ace.WaitingTime* が 3,000 msec では，それぞれのクラッシュ時間順に，約 27,000 msec，約 17,000 msec そして約 7,000 msec であった．また，*Ace.WaitingTime* が 6,000 msec では，約 54,000 msec，約 34,000 msec そして約 14,000 msec であった．クラッシュの検出にメッセージのタイムアウトを用いるため，回復時間は式 (1) となっているのが分かる．

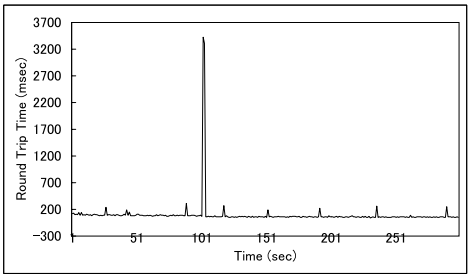


図 12 オーバヘッド
Fig. 12 Overhead.

5.3 再配置にともなうオーバヘッド

アプリケーションを継続して利用できることを確認するために，実行中に再配置を行い，その間の参加ノードと調停ノード間の RTT 値を 1 sec 間隔で計測した．また，メトリックにネットワーク接続時間を用い，パラメータは *Ace.WorkerPeriod* を 60,000 msec に，*Ace.WaitingTime* を 3,000 msec に，*Ace.RetransmitMessageNum* を 0 回に設定した．実験開始 12,000 msec 後に調停ノードをクラッシュさせ再配置を行った．**図 12** は，横軸が計測時間，縦軸が RTT 値を表している．図 12 から，約 75 msec，クラシュ直後は約 3 sec であるのが分かる．これは，クラッシュ直後に送信したメッセージが調停ノードに受信されず，新規調停ノードに再送されたため，その間に要

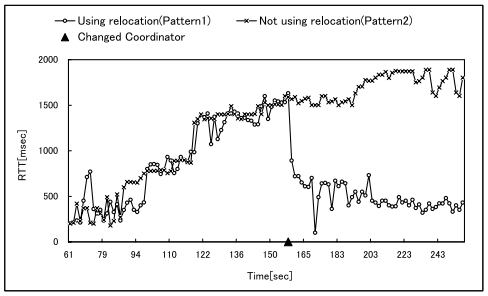


図 13 適応性
Fig. 13 Adaptability.

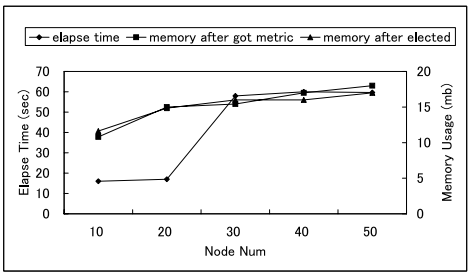
した時間を表している．ACE ではメッセージキューによって再配置が発生した場合でもアプリケーションに透過的に調停ノードを再配置できることが分かった．

5.4 適 応 性

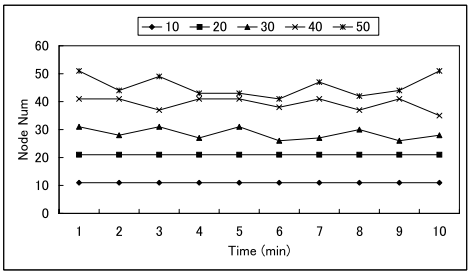
ノードの環境に適応して最適な調停ノードを決定することを実験により評価した．実験では，環境メトリックとして端末処理能力を用い，制約条件として CPU 使用率が 90%以上（*Metric*.1.*Threshold* ≥ 90）になった場合に再配置を行うように設定した．また，パラメータは *Ace.WorkerPeriod* を 60,000 msec に，*Ace.WaitingTime* を 3,000 msec に，*Ace.RetransmitMessageNum* を 0 回に設定した．**図 13** は，調停ノードの負荷を高くし，再配置をした場合としない場合の調停ノード間の RTT 値の推移を示している．実験では，初回の調停ノード選任で NodeA が調停ノードとなった．再配置をした場合，開始直後から負荷を高くするまでの平均 RTT 値は 371 msec，100 sec 後に NodeA の負荷を高くした以降は 1,206 msec，そして再配置してから実験終了までは 490 msec であった．つまり，負荷が高くなると約 81%上昇したが，再配置により NodeB が新規調停ノードになると，負荷を高くしたときに比べ約 61%減少した．一方，再配置をしない場合，開始直後から負荷が高くなるまでの平均 RTT 値は 413 msec，100 sec 後に NodeA の負荷を高くした以降は 1,487 msec になり約 90%上昇している．これにより，再配置によって平均 RTT 値が 997 msec 短縮され，ノードの環境情報に適応していることが分かる．

5.5 スケーラビリティ

調停ノードに対する負荷テストによって，スケーラビリティの評価を行った．実験では，J2SE1.4.2 および Project JXTA の bench プロジェクト[7] のベンチマークツールを用いて，1 台の端末に最大 20 ノードを起動し，計 50 ノードを調停ノードに接続した．各 ACE ノードのパラメータは *Ace.WorkerPeriod* を 60,000 msec に，*Ace.WaitingTime* を 3,000 msec に，*Ace.RetransmitMessageNum* を 0 回に設定した．**図 14** (a) は，横軸にノード数，縦軸（左）に経過時間，縦軸（右）にメモリ使用量を表している．経過時間は，最初のノードからメトリックを受信してから調停ノードを選任するまでの平均時間で，メモリ使用量は，メトリックを受信したときと調停ノードを選任したときの平均メモリ使用量である．20 ノードを超えると，選任時間が大幅に増えていることが分かる．これは，複数のノードから同時にメッセージを受信したときに，JXTA のメッセージ処理に時間がかかっていると考えられる．また，メモリ使用量に関しては，各ノード数においてそれほど変化は見られず，1 ノードあたり 400 kb から 1,000 kb 程度の使用量となり，VM の標準メモリの量（64 MB）でも十分であるといえる．図 14 (b) に，調停ノードが一定間隔ごとに受信したメトリックのノード数を示す．横軸に経過時間，縦軸にノード数を表している．30 ノード以上になると，すべてのノードからメトリックを受信していないことが分かる．これは，図 14 (a) で明らかなように，30 ノード以上の場合，調停ノードの選任時間に 60 秒程度かかっているため，選任時間が経過し，すべてのノードからメトリックを受信できなかったと考えられる．これは，JXTA のメッセージキューの最大サイズを変更するか，*Ace.WorkerPeriod* を 60,000 msec 以上に設定することで解決することができると考えられ



(a) 経過時間



(b) 受信ノード数

図 14 スケーラビリティ
Fig. 14 Scalability.

る．本評価により，標準の VM および JXTA の設定では，20 ノード程度であれば十分利用可能であり，また，JXTA や ACE プロパティ（選任間隔や利用メトリック）の設定を変更することで，これ以上のノード数でも対応可能であると考えられる．

6. おわりに

本論文では，P2P 環境において，動的に調停ノードを決定し，必要に応じて再配置する ACE プラットフォームを提案した．P2P ネットワークは，実ネットワークや端末の種類が異なるヘテロジニアスな環境上に形成されているため，任意に調停ノードを決定することはできない．しかし，ACE では，ノードの環境情報やポリシ定義を用いることで，アプリケーションの目的に応じた，最適な調停ノードを決定することが可能になる．ACE は JXTA を用いて，ミドルウェアとして実装し，API を利用したサンプルアプリケーションと実験による評価によって，ACE アーキテクチャの有効性を示した．

今後の課題として，3.5 節で説明したように，競合時の total ordering の保証，タイムアウトによる障害検出の不正確性を改善する必要がある．また，グループの中で唯一の調停ノードの選択は，実験により負荷が集中していることが分かり，ノード数の規模拡大に対応することが難しことが分かった．そこで，スケーラブルな調停ノードの選択手法を明らかにする必要がある．

謝辞 本研究は早稲田大学特定課題 2004*A*-327, NTT DoCoMo との共同研究として行われました．また，本研究の一部は，*JGN*2（*JGN*2-*A*16065）のサポートにより行われました．

参 考 文 献

1) Yoshinaga, H., Tsuchiya, T. and Koyanagi, K.: A Coordinator Election Using the Object Model in P2P Networks, *3rd International Workshop on Agents and Peer-to-Peer Computing*, New York City (July 2004).
2) Yoshinaga, H., Tsuchiya, T. and Koyanagi, K.: Relocation of Mediation Node for Data Synchronization and Consistency in P2P Collaborative Applications, *KEST2004*, Beijine (Sep. 2004).
3) Tsuchiya, T., Yoshinaga, H., and Koyanagi, K.: STARCast: Streaming Collaboration on heterogeneousEnvironment anywhere, *ACM Multimedia 2004 workshop*, New York (Oct. 2004).
4) Tsuchiya, T., Yoshinaga, H., and Koyanagi, K.: Streaming Collaboration on Heterogeneous Transport Networks, *KEST2004*, Beijine (Sep. 2004).
5) JXTA. http://www.jxta.org/
6) JXCube. http://jxcube.jxta.org/
7) bench. http://bench.jxta.org/
8) Bernard, T., Ahkil, A., Mohamed, A., Mike, D., Carl, H., Jean-Christophe, H., Eric, P. and Bill, Y.: Project JXTA 2.0 Super-Peer Virtual Network (May 2003). http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf
9) 高杉耕一，片山 穣，久保田稔，小柳恵一：ノードの移動性を実現するモビリティ技術，情報処理学会論文誌，Vol.42, No.7, pp.1828–1839 (2001).
10) The Gnutella protocol speci.cation (2000). http://dss.clip2.com/GnutellaProtocol04.pdf
11) Clarke, I., Sandberg, O., Wiley, B. and Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system, *Workshop on Design Issues in Anonymity and Unobservability*, pp.311–320, ICSI, Berkeley, CA, USA (July 2000).
12) Groove. http://www.groove.net/
13) skype. http://www.skype.com/
14) 石川貴士，石原 進，井手口哲夫，水野忠則：遅延差のあるネットワークにおけるメンバ間公平性保証方式の特性評価，情報処理学会論文誌，Vol.42, No.7, pp.1817–1827 (2001).
15) Abu-Amara, H.: Fault-tolerant distributed algorithms for election in complete networks, *IEEE Trans. Comput.*, C-37, pp.449–453 (1988).
16) Afek, Y. and Gafni, E.: Time and message bounds for election in synchronous and asynchronous complete networks, *SIAM Journal on Computing*, 20, pp.376–394 (1991).
17) Hatzis, K., Pentaris, G., Spirakis, P., Tampakas, V. and Tan, R.: Fundamental Control Algorithms in Mobile Networks, *Proc. 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp.251–260 (1999).
18) Malpani, N., Welch, J. and Vaidya, N.: Leader Election Algorithms for Mobile Ad Hoc Networks, *4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Boston, MA (Aug. 2000).
19) 鈴木貴也，石原 進，水野忠則：アドホックネットワークにおけるトポロジ依存機能の動的再配置，情報処理学会論文誌，Vol.43, No.12, pp.3959–3969 (2002).
20) Schooler, E.M., Manohar, R. and Chandy,

K.M.: An Analysis of Leader Election for Multicast Groups, Technical Report, AT&T Labs-Research, Menlo Park, CA (Feb. 2002).
21) Singh, S. and Kurose, J.F.: Electing 'good' leaders (election leader algorithm), *Journal of Parallel and Distributed Computing*, Vol.21, No.2, pp.184–201 (May 1994).
22) Rampath, S. and Dahbura, A.: A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies, *IEEE Trans. Comput.*, Vol.44, No.4 (1995).
23) Duarte Jr, L. and Nanya, T.: A Hierarchical Adaptative Dist. System-level DiagnosisAlgorithm, *IEEE Trans. Comput.*, Vol.47, No.1 (1998).
24) Lamport, L., Shostak, R. and Pease, M.: The Byzantine Generals Problem, *ACM Trans. Prog. Lang. Syst.*, Vol.4, No.3, pp.382–401 (1982).
25) Cristian, F.: Reaching Agreement on Processor-group Membership in Synchronous Distributed Systems, *Distributed Comp.*, 4, pp.175–187 (1991).
26) Birman, K.: The Process Group Approach to Reliable Distributed Computing, *Comm. ACM*, Vol.9, No.12. pp.36–53 (1993).
27) Chandra, T. and Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems, *J. ACM*, Vol.43, No.2, pp.225–267 (1996).
28) Andrew, T. and Maarten, S.: *Distributed Systems: Principles and Paradigms*, 1st edition, Prentice Hall (2002).
29) Elnozahy, E., Johnson, D. and Wang, Y.: A Survey of Rollback-Recovery Protocols in Message-Passing Systems, Technical Report CMU-CS-96-181, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (Oct. 1996).

(平成 16 年 12 月 10 日受付)
(平成 17 年 6 月 9 日採録)

吉永　浩和（学生会員）
2000 年獨協大学経済学部卒業，ソフトウェアエンジニアとして勤務，2005 年早稲田大学大学院情報生産システム研究科修士課程修了．現在，早稲田大学大学院情報生産システム研究科博士課程在学中．分散システム，P2P システムに関する研究に従事．電子情報通信学会学生員．

土屋　　健（学生会員）
2001 年東京理科大学理学部卒業，2003 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，早稲田大学院情報生産システム研究科博士課程在学中．分散マルチメディアシステムに関する研究に従事．電子情報通信学会学生員．

小柳　惠一（正会員）
1975 年慶應義塾大学工学部電気工学科卒業．1977 年同大学大学院修士課修了．1998 年大阪大学大学院通信工学専攻博士課程修了．1977 年日本電信電話公社（現 NTT）入社．1999 年 NTT 未来ねっと研究所部長．2003 年早稲田大学大学院情報生産システム研究科教授．ネットワークミドルウェア，知能化ネットワークの研究開発に従事．博士（工学）．1996 年度電気通信普及財団賞受賞．IEEE Senior Member，ACM 会員．