集中管理された利用者用計算機の利便性向上に関する研究

髙田 真吾

システム情報工学研究科
筑波大学

2013年3月

概要

本論文では、多数の計算機と多数の利用者を持つ計算機システムにおける、集中管理された利用者用計算機の利便性を向上させる方法について述べる。現在の集中管理された計算機システムでは、その利用者の層が広くなってきているにもかかわらず、集中管理の仕組みがその変化に対応できていない。そのため、情報リテラシや計算機利用のスキルの高い利用者であってもそれらの低い利用者に向けた計算機を利用しなければならないことが起こる。これは計算機システム全体としての利便性を低下させてしまう。

本研究では、集中管理された利用者用計算機において利用者認証を強化することで計算機の利便性を向上させることを提案する。提案方式では、利用者認証を強化することで利用者のスキルに応じて計算機を利用可能とする。これにより、幅広い利用者層を持つ計算機システムであったとしても利便性を向上させることを可能にする。本論文では集中管理された利用者用計算機の事例として、大学における重要な計算機である実習室に配置された計算機と図書館における蔵書検索用の端末を取り上げ、提案方式を評価する。

まず、大学の計算機実習室に配置された計算機について考える。本研究では、スキルの高い利用者に対して、より高い権限を持った OS を提供できるようにするため、集中管理された計算機上で、利用者と計算機の組み合わせに応じて OS の起動・終了を制御する仕組みを提案する。この仕組みにより、管理者が認めた情報リテラシや計算機利用のスキルの高い利用者に対して、より自由度の高い OS を提供することを可能とする。そのような仕組みを持つ起動・終了制御システムを実装し、利用者と計算機の組み合わせに応じた起動・終了制御を実現できることを示す。ここで、OS を提供する方法の一つであるネットワークブートシステムでは、OS の配信処理がボトルネックとなる。本研究では、このボトルネックを P2P 方式のディスクイメージ配信システムにより解消する。先行研究として、KNOPPIX を P2P 方式で配信する研究があるが、本研究では、実習室のような環境を対象とするため、多数の OS を配信可能とする。また、実際の実習室においては、その起動分布は正規分布に従うことから、そのような場合に起動を完了するまでにかかる時間がどのように変化するかを示す。多数の OS を配信可能とする P2P 方式のディスクイメージ配信システムを設計、実装し、正規分布に従って起動処理を開始する場合において、従来の NFS による方法と、P2P による方法での起動を完了するまでにかかる時間を測定する。その結果から、正規分布に従った起動分布においても、P2P 方式を用いることで配信サーバのトラフィックを軽減し、起動を完了するまでにかかる時間を短縮できることを示す。

次に、大学図書館における蔵書検索用の端末において、利用者認証を強化することにより利便性を向上させることについて述べる。利用者認証を強化することにより、大学の構成員は蔵書検索以外の目的にも端末を利用できるようになり利便性が向上する。その際に使用するアカウントとして、管理者が発行したユーザアカウントだけでなく、多数の認証システムを用いて認証を受けることができれば、より多くの来館者が Web アクセスを行うことが可能となる。本研究では、このような認証とアクセス制御を行う仕組みを提案する。この仕組みにより、利用者は自らが持つアカウントを用いて認証を行い、そのアカウントから提供される属性情報に基づいたアクセス制御ルールの下で Web アクセスを行うことを可能とする。そのようなアクセス制御システムを実装し、図書館の蔵書検索端末に適用し、その有用性を示す。

目 次

図 目 次

表 目 次

第1章　はじめに

本論文は、多数の計算機と多数の利用者を持つ計算機システムにおける、集中管理された利用者用計算機の利便性を向上させる方法について述べたものである。

企業や大学のように、多数の計算機と多数の利用者を持つ組織における計算機システムでは、計算機システムの提供目的を満たすことができるような状況を維持することが求められる。管理者はこれを実現するために、多数の計算機を効率良く管理下に置くことができるよう、計算機システムの集中管理を行う。たとえば、利用者のアカウントを集中的に管理したり、利用者が操作する計算機 (**利用者用計算機**) へのアプリケーションのインストールやパッチの適用といった作業を一元化したりする。このような方法を用いることにより、多数の計算機を管理下に置くことを容易にしている。

しかし、このような集中管理の方法を取り入れる場合、全体として利便性が低下してしまうことが起こる。たとえば、筑波大学の全学計算機システムは、学生や教員と言った構成員に対して多数の計算機を提供する。平成 22 年度では、約 1,000 台の計算機を、約 21,000 人を超える利用者に提供するが、その計算機上で利用可能な OS は、管理者が集中管理を行うことができるように設定された 2 種類の OS のみに限定されている。また、利用者はその OS 上で制限付きのユーザとしてログインし、あらかじめインストールされているアプリケーションを使うことになっている。情報リテラシや計算機利用のスキル (以下本論文では**スキル**) の高い利用者は、制限付きのユーザでは普段使い慣れた操作、たとえば、コマンドプロンプトが使えない等の不便を感じる。また、特に教員から見た場合、パケットキャプチャのような、管理者権限が必要となるアプリケーションを動作させるような実験を学生を対象に提供したい場合も考えられる。このような場合、教員は、集中管理の仕組みを利用できないので、自ら CD-ROM を作成する必要があるなど、不便を感じる。

また、ソフトウェアの利用だけでなく、通信についても管理上の理由による利便性の低下が見られる。たとえば、筑波大学附属図書館では、サービスの一環としてオンライン蔵書検索システム (OPAC) が提供されている。図書館の来館者は、OPAC 端末を用い、蔵書検索や図書館の Web ページの閲覧などを行うことができる。この OPAC 端末は、来館者が蔵書情報を検索するために設置されているものである。図書館の OPAC 端末は、大学の学生や教職員のような正規の構成員の他に来館者が利用することがある。来館者がこの端末を用いて外部の任意の Web サイトを閲覧した場合、不正利用などのインシデントが生じる可能性がある。そのような不正利用を防ぐために、図書館の OPAC 端末は利用できる Web サイトを制限することが一般的である。そうなると、大学の学生や教職員のような正規の構成員は不便を感じる。

このように、従来の集中管理された利用者用計算機は、集中管理されていない利用者用計算機よりもしばしば利便性が低下する。

本研究では、集中管理された利用者用計算機において利用者認証を強化することで計算機の利便性を向上させることを提案する。本研究で提案する方法では、利用者認証を強化することで、集中管理された利用者用計算機であっても利用者のスキルに応じて利用可能とする。これにより、幅広い利用者層を持つ集中管理された計算機であっても、高いスキ

ルを持つ利用者にとって利便性を向上させることを可能にする。本論文では集中管理された利用者用計算機の例として、大学における重要な計算機である以下の 2 つを取り上げる。

- 計算機実習室に配置された計算機
- 図書館における蔵書検索用の端末

これらの利用者用計算機に対して利用者認証を強化することで、計算機の利便性を向上させることが可能であることを示す。

まず、大学の計算機実習室に配置された計算機について考える。本研究では、スキルの高い利用者に対して、より高い権限を持った OS を提供できるようにするため、利用者と計算機の双方を識別することにより柔軟に OS を配信することができる仕組み [59] を提案する。この仕組みでは、管理者が「利用者と計算機の組み合わせ」を特定することができるため、管理者が認めたスキルの高い利用者に対して、より自由度の高い OS を実行することを許可することができるようになる。これにより、従来はそのための CD-ROM を作成して対応していたようなパケットキャプチャ実験の場合などに対しても、集中管理された計算機を用いて対応することができ、利便性を向上させることができる。

この「利用者と計算機の組み合わせ」を識別する方法として、本研究では利用者の識別には USB トークンや IC カードといった認証デバイスを、計算機の識別には TPM (Trusted Platform Module)[9] のようなハードウェアモジュールを、それぞれ利用する。セキュア VM を実現するための仮想計算機モニタである BitVisor[51] では、政府機関で利用することを想定し、セキュリティを強化するため、IC カード (国家公務員カード) を認証に用いる。同様に、集中管理された利用者用計算機の管理者は、通常のユーザ名とパスワードによる認証では利用者に高い権限を与えるには不十分であると考えることがある。そのような場合、管理者は、IC カードや USB トークンといった偽造が困難な認証デバイスを用いて利用者認証を強化する。また、管理者は、任意の集中管理された利用者用計算機において高い権限を与えるのではなく、特定の計算機に厳密に制限したいこともある。そのような場合、管理者は、TPM といったハードウェアを用いて計算機を制限したいと考えることがある。このように管理者はハードウェアにより利用者と計算機の認証を強化し、その利用者と計算機の組み合わせに応じて通常よりも高い権限を持った OS を利用可能にしたいと考えることがある。本研究では、そのような管理者の要求に応えるような OS の起動・終了制御を行うシステムを設計し実装した [59]。

利用者と計算機の双方の識別を行う上での技術的課題として、その利用者がその計算機を利用していることをどのようにして保証するか、というものがある。認証デバイスや TPM などには、利用者や計算機を識別する情報として、証明書を格納することができる。この証明書を用いて認証を行う方法としては、HTTPS で利用される、SSL(Secure Socket Layer) のクライアント認証がある。しかしこの方法では、認証において同時に使うことができる証明書は高々 1 つであり、利用者の証明書と計算機の証明書の双方を同時に認証するには不十分である。本研究ではこの問題の解決のために、計算機に固有の証明書を用いて HTTPS クライアント認証セッションを構築し、その上で利用者に固有の証明書を用いたチャレンジアンドレスポンス方式の認証を行う。これにより、その利用者がその計算機上で認証処理を実行していることを確実なものとすることができる。

このようにして利用者と計算機の双方を識別し、その組み合わせに応じて起動・終了制御を行うシステムを設計、実装した [59]。その結果、利用者と計算機の組み合わせに応じて柔軟に OS を配信することができることを示した。しかし、課題として、計算機上で動作させる OS のディスクイメージの配信に時間がかかるという問題がある。

これはネットワーク経由で OS を配信する、いわゆるネットワークブートシステムに共通の問題である。多数の利用者に対して、大容量の OS イメージを配信する必要があるため、配信を行うサーバやネットワークがボトルネックになってしまうという問題である。これを解決する方法として、その配信を Peer-to-Peer(P2P) 方式で行うものがある。

そのような先行研究として、KNOPPIX を用いたシンクライアント環境を P2P 方式で配信するもの [66] がある。この手法では、配信対象を Linux(KNOPPIX) とし、その起動プロセスに手を加えることにより、起動処理の段階に応じたツリーを構成し、配信の効率化を行っている。本研究は、大学の実習室のような環境を対象とし、多数の計算機と多数の利用者を対象に利用者と計算機の組み合わせに応じて多数の OS を配信することを目指す。そのため、多数の計算機に対し、KNOPPIX だけでなく Microsoft Windows のようなプロプライエタリな OS を含めて多数の OS を配信可能であることが必要である。加えて、計算機を授業で利用するような場合、授業開始時や教員の指示などをきっかけに、多数の計算機が起動処理を開始するといったことが起こる。この場合、配信サーバに起動要求が集中してしまい、全ての計算機が起動を完了するまでに長い時間がかかってしまうことが考えられる。そのような事態に陥ってしまうと授業の進行に影響を与えてしまうため、本研究の対象となる環境においては、起動要求の集中に対してどのように振る舞うかが重要となる。特に授業の例では人間が計算機の起動を行うことから、1 秒程度の間に全ての計算機が起動を開始するのではなく、ある時刻を基準に正規分布に従ってそれぞれの計算機が起動を開始する。そのような場合において、授業を滞らせることなく、OS の配信を行えることが必要である。

本研究では、これらの要求を満たす P2P 方式のディスクイメージ配信システムを設計、実装、評価した [60]。特に、実習室のような環境において起こりうる、多数の計算機が正規分布に従った分布で起動を開始するような場合について評価を行った。その結果、正規分布に従う起動分布であっても、従来の NFS による方法に比べ、起動にかかる時間が短縮されることを示した。また、ワーストケースである一斉起動の場合でも、従来の NFS による方法に比べ、P2P 方式の方が早く起動を完了することを示した。これにより、実際の運用において起こりうる起動分布においても、その起動所要時間を短縮し、より速やかに OS を利用可能とすることができることを示した。

次に、図書館における蔵書検索用の端末における認証について考える。この端末は、Web 上で提供される蔵書検索サービスを来館者に提供するために設置されるものであり、来館者は認証を行うことなく、蔵書検索サービスや図書館の Web ページの閲覧を行うことができる。この端末を用いて、認証を受けることで任意の Web ページも閲覧できるようになれば、利便性は向上する。

来館者が組織の構成員であれば、その組織のユーザアカウントを用いて認証を受けることができる。そうでない場合には、来館者が任意の Web ページを閲覧できるようにするためには、一時的アカウントが必要となる。この一時的アカウントは、管理者から見ても利用者から見ても、その管理が煩雑となってしまう問題がある。一時的アカウントを用いることにより、管理者も利用者も、管理すべきアカウントが一つ増えてしまう。

もし一時的アカウントを発行してもらうかわりに、来館者が既に持っているアカウントを用いて認証を受けることができれば、この問題を解決することができる。たとえば、管理者が Facebook による認証を認めている場合、この来館者が既に Facebook のアカウントを持っていれば、そのアカウントを用いて認証を受けることができる。これにより、管理者は一時的アカウントを発行する必要がなくなり、来館者は新たなアカウントを管理しなくて済む。来館者が利用可能な認証システムの数が多ければ、それだけ一時的アカウント

の発行を不要とすることができる。

Web アクセス時の認証として、多数の認証システムに対応したアクセス制御を行うためには、認証システムから得られる利用者の情報 (**属性情報**) を、認証システムに依存せずに取り扱えることが必要となる。また、新たな認証システムが登場しても容易に対応可能であることが望ましい。これを実現する上での技術的課題として、利用者の識別と属性情報の取り扱いがある。

Web アクセス時に利用者に認証に要求する場合において、多数の認証システムを利用可能とする方法として SPNEGO[65] がある。SPNEGO では、GSS-API[42] に対応した認証システムであれば、その中から望ましいものを選び、認証に用いることができる。しかし、この SPNEGO を使う場合、対象としたい認証システムの認証インタフェースを、GSS-API のインタフェースにあわせる必要がある。また、SPNEGO の場合、新たな認証システムに対応するためには、Web ブラウザ側と Web プロクシサーバ側のモジュールを実装し、その間で行われる通信のプロトコルを定める必要があり、それらのモジュールのインストールが必要となってしまうという問題がある。

そこで本研究では、**認証方法に依存しないユーザ識別子 (MI-UID)** と、その識別子を用いて属性情報を格納する**ユーザ属性ストア**を用いることにより、認証システムに依存せずに属性情報を取り扱うことを可能とし、多数の認証システムを利用可能なアクセス制御を実現する [61]。認証方法に依存しないユーザ識別子とは、利用する認証システムによらずにユーザを識別できるものである必要がある。そのようなユーザ識別子として、ブラウザ側に手を加えることにより送出するユーザに固有の文字列や、詐称を防ぐことができる環境における IP アドレスのような情報を利用する。この認証方法に依存しないユーザ識別子の要件については、詳しくは第 4 章で述べる。この認証方法に依存しないユーザ識別子をキーとして、そのユーザの属性情報をユーザ属性ストアに格納する。これにより、新たな認証システムに対応する場合においても、認証システムに依存しない属性情報の取り扱いを可能とし、多数の認証システムに対し容易に対応することを可能とする。

本研究では、このような多数の認証システムを利用することが可能な Web プロクシによるアクセス制御システムを設計、実装、評価した。認証システムとして Shibboleth に対応したアクセス制御システムを、筑波大学附属図書館の 182 台の OPAC 端末に適用し、1 年以上運用を行いその有用性を示した。また、それ以外の認証システムとして、Kerberos[52] のような既存のプロトコルや、Facebook のような近年になって登場した認証システムにおいても、容易に対応可能であることを示した。

本論文の構成は次のようになっている。第 2 章では、集中管理された利用者用計算機における、利用者と計算機の組み合わせに応じた OS の起動・終了制御システムについて述べる。第 3 章では、ネットワークブートシステムにおける、P2P 方式のディスクイメージ配信システムについて述べる。第 4 章では、多数の認証システムを利用可能な URL レベル外向きアクセス制御システムについて述べる。最後に、第 5 章で本論文をまとめる。

第2章　利用者と計算機の組み合わせに応じたOSの起動・終了制御

本章では、集中管理された利用者用計算機のうち、大学の計算機実習室に配置された計算機において利用者認証を強化することにより利便性を向上させる手法について述べる。これを実現するために、利用者と計算機の双方を識別し、その組み合わせに応じてOSの起動・終了制御を行う仕組みを提案する。これにより、管理者の認めた、情報リテラシや計算機利用のスキル (**スキル**) の高い利用者に対して、より高い権限を行使できるようなOSを配信することを可能とし、そのような利用者の利便性を向上させることができる。なお、本論文では、単にOSと言った場合、狭義のオペレーティングシステムだけでなく、その上で利用可能なアプリケーションを含めたものを意味する。

多数の計算機を多数の利用者に対して提供する環境においては、それらの計算機を適切な管理下に置くことが重要である。筑波大学の例として、平成22年度では、約21,000人の利用者に対し約1,000台の計算機を提供している。このようなシステムを適切に管理するため、管理者は集中管理を行っている。大学の計算機システムの場合、そのシステムの利用者のスキルは広く分布している。そのため、管理者はスキルの低い利用者にあわせて計算機システムを配備している。具体的には、利用者は管理者が管理しているOSを利用し、そこに管理者権限を持たない制限付きのユーザとしてログインし、管理者によりあらかじめインストールされたアプリケーションを利用する。このような環境を提供することにより、スキルの低い利用者が管理者によりインストールされたアプリケーションを誤って削除してしまったり、他の利用者に影響を与えるような操作を行ってしまったりすることを防いでいる。その一方で、スキルの高い利用者がパケットキャプチャのような管理者権限を必要とするような作業を行う場合には、集中管理の仕組みを利用できないので、自らCD-ROMを作成する必要があるなど、不便を感じる。

本研究では、利用者と計算機の双方を識別することにより柔軟にOSを配信することができる仕組みを提案し、この問題を解決する。管理者が「利用者と計算機の組み合わせ」を特定できる仕組みにより、管理者が認めた利用者が、管理者が認めた計算機上で、より自由度の高いOSを実行することを許可することができるようになる。これにより、スキルの高い利用者であると管理者が認めた場合に、CD-ROMを作成するなどの作業なしに、特定の計算機上でのみ特定のOSの利用を許可することができる。

セキュアVMを実現するための仮想計算機モニタであるBitVisor[51]では、政府機関で利用することを想定し、セキュリティを強化するためにICカード (国家公務員カード) を認証に用いる。この偽造が困難なICカードを用いて、利用者を識別する。同様に、集中管理された利用者用計算機の管理者は、通常のユーザ名とパスワードによる認証では利用者に高い権限を与えるには不十分であると考えることがある。そのような場合、管理者は、ICカードやUSBトークンといった偽造が困難な認証デバイスを用いて利用者認証を強化する。また、管理者は、任意の集中管理された利用者用計算機において高い権限を与えるのではなく、特定の計算機に厳密に制限したいこともある。そのような場合、管理者は、TPM (Trusted Platform Module) といったハードウェアを用いて計算機を制限したいと

考えることがある。このように管理者はハードウェアにより利用者と計算機の認証を強化し、その利用者と計算機の組み合わせに応じて通常よりも高い権限を持った OS を利用可能にしたいと考えることがある。本研究では、そのような管理者の要求に応えるような OS の起動・終了制御を行うシステムを提案する。

このように本研究の手法では、認証デバイスと TPM を用いて利用者と計算機の双方を識別し、その組み合わせに応じて起動可能な OS を決定する。認証デバイスに格納された証明書を用いて認証を行う方法としては、HTTPS クライアント認証がある。HTTPS クライアント認証では、クライアントがサーバに対して証明書を提示し、サーバがそれを検証することにより、認証を行うことができる。しかし、従来の方法では、クライアントが持つ 2 種類の証明書を同時に用いて認証を行うことができないため、その利用者が、その計算機を用いて、認証処理を行っていることを確実なものとすることができない。利用者と計算機の組み合わせに応じた制御を行う上では、これを実現する必要がある。

そこで本研究では、HTTPS クライアント認証と、そのセッション上で行われるチャレンジアンドレスポンス方式の認証を組み合わせて用いる。本研究の手法では、利用者と計算機の双方の認証処理は、認証デバイスと TPM のそれぞれに格納された証明書を用いて行われる。まず計算機の TPM に格納された計算機に固有な証明書を用いて、認証サーバとの間で HTTPS クライアント認証セッションを確立する。このセッション上で、認証サーバが生成したチャレンジデータと利用者に固有な証明書を用いて、チャレンジアンドレスポンス方式で利用者の認証を行う。これにより、計算機と利用者の双方を同時かつ確実に識別することを可能とする。この計算機と利用者の組み合わせに応じて、認証サーバは起動可能な OS を制御することができる。このような手法を用いることにより、本研究では、この利用者と計算機の組み合わせに応じて管理者が設定したルールに基づいて起動可能な OS を制御することを可能とする。

このような起動・終了制御システムを設計・実装し、その有用性を評価した。その結果、利用者と計算機の組み合わせに応じて起動可能な OS を制御できることを示した。

2.1 関連研究

本節では、OS の起動や終了を制御するための既存の手法について述べる。

2.1.1 PXE –Preboot eXecution Environment–

クライアントコンピュータには必要最小限のプログラムだけを配置し、起動に必要なファイルをネットワーク経由で取得することにより、ディスクレスブートを実現するための規格として、Intel 社が策定した PXE[30] がある。PXE では、NBP (Network Boot Program) を TFTP を用いて取得し、この NBP が OS を起動する。利用者に対して複数の起動候補を提示、選択させることが可能であるほか、MAC アドレスに応じて起動する OS を変更することも可能である。

PXE のような手法では、提供する環境のバリエーションの数だけ利用者に選択肢を提供するため、利用者はその中から適切な環境を選択する必要がある。これは煩雑な手間であり、また利用者が誤って終了方法を知らない OS を起動してしまうといった操作ミスが起こりうる。また、その OS を起動することそのものを制御したいといったことを行うことができない。加えて、PXE はブートローダのように OS の起動のみを制御するため、OS の動作中や起動後、OS の終了後に何かの処理を行うといったことができない。

本研究の手法では、認証デバイスとハードウェアの組み合わせから起動する OS を決定するため、利用者が利用できない OS が起動することは起こらない。また、本研究の手法では OS の起動後も OS から独立して継続的に動作するため、OS の終了まで制御を行うことが可能である。

2.1.2 起動処理におけるコンポーネント検証

電源投入直後から OS の起動までの間に処理が行われるコンポーネントの状態値を検証することにより、不正なコンポーネントの挿入や改竄といった脅威からシステムを保護し、起動処理の完全性を検証可能とする試みとして、次のようなものがある。

AEGIS[8] は、拡張ボードに状態値を格納しておき、起動時にコンポーネントの状態値を比較することでシステムの完全性を検証する。sAEGIS[33] はこの AEGIS をさらに拡張したもので、ブートローダとして grub をサポートしたことにより起動できる OS の種類が増え、また状態値の保存にスマートカードを採用したことにより、システムの完全性のより高度な検証を実現している。

Intel TXT[31] は、システムの完全性の検証を CPU や TPM を用いて実現する実装である。CD や DVD などのメディアに起動に必要なファイルだけを格納しておき、それ以外に必要となるファイルをネットワーク経由で取得することで OS を起動するシステムとして、HTTP-FUSE KNOPPIX[54] がある。中村ら [44] は、この HTTP-FUSE KNOPPIX を TPM を搭載したコンピュータ上で利用する際に、システムの完全性を記録する手法を提案、実装している。この手法では、TPM にコンポーネントの状態値を記録することで、システムの完全性を検証することを可能としている。また、ディスクイメージを構成するブロックファイルを取得する際に利用するサーバが信頼できるものであるかどうかはサーバが持つ証明書を用いて検証し、取得したファイルについても、ハッシュ値を用いてそのブロックファイルに改竄が行われていないかを検証している。

これらの研究は、起動に必要なコンポーネントが想定した状態で存在していることを検証することを可能としている。この仕組みを利用して、特定の認証デバイスが挿入されていなければ OS を起動することができないようなシステムを構築することは可能であるが、OS の起動後に認証デバイスが抜かれるというハードウェア構成の変更を検出するためには起動する OS に監視を行うような変更を加える必要がある。

それに対し、本研究の手法は、OS の起動後も認証デバイスが挿入された状態が継続していることを監視する。これにより、特定の認証デバイスと特定のハードウェアの組み合わせでのみ許可された OS が稼動することを保証するだけでなく、起動する OS に変更を加えることなく、OS の起動後もその組み合わせに変更がないことを保証している。

2.1.3 ハードウェアの固有情報による実行制御

システムのハードウェアが持っている固有情報を用いることにより、特定のハードウェア上での特定のアプリケーションの動作を制御する仕組みを実現することができる。

XOM[41] は CPU が固有の鍵を持つことでこの仕組みを実現しているアーキテクチャである。この CPU 上で動作するプログラムをあらかじめ CPU の公開鍵で暗号化しておくことで、プログラムがその CPU 上でのみ動作するということが実現可能となる。これにより、OS が稼動する計算機 (CPU) を制限する仕組みを実現することができる。

これに対し、本研究の手法では、認証デバイスとハードウェア固有情報の組み合わせにより、OS の稼動を制御する。さらに、ハードウェアと認証デバイスの組み合わせが異なると、異なる OS が起動するように制御可能である。

XOM では、このアーキテクチャ用に OS やアプリケーションを変更する必要があるが、本研究の手法では、アーキテクチャに変更の必要がないため、既存の OS に変更を加えることなく起動制御を行うことができる。

2.1.4 BitVisor

BitVisor[51] は、政府機関で利用することを想定したセキュア VM を実現するための機能を持つ仮想計算機モニタである。そのため、この BitVisor では、IC カード (国家公務員カード) を用いて利用者を識別する機能を持つ。より高いセキュリティを実現する必要がある場合には、IC カードに格納された鍵を用いて計算機のストレージに格納されるデータを暗号化することができる。この機能を用いることにより、対応する鍵が格納されている IC カードの所有者のみがストレージに格納された OS を起動することができるような仕組みを実現することができる。

BitVisor において IC カードで利用者の識別を行うのと同様に、集中管理された利用者用計算機の管理者は、通常よりも高い権限を持った OS を提供するにあたり、利用者をより厳密に識別したいと考えることがある。そのような場合では、利用者の識別に、IC カードや USB トークンと言った偽造が困難な認証デバイスを用いて利用者認証を強化する。計算機についても同様に、TPM のようなハードウェアを用いてその認証を強化する。これにより、管理者は認証デバイスと TPM を用いて利用者と計算機の双方を識別することが可能となり、その組み合わせに応じて、より自由度の高い OS を起動することを許可することができる。また、本研究の手法は、ストレージに格納された OS だけでなく、ネットワーク経由で配信される OS についても起動・終了制御対象とすることができる。

2.2 起動・終了制御システムの設計

本研究で提案する、OS 制御レイヤと認証デバイスを用いた柔軟な OS の起動・終了制御システムは、以下の要件を満たす。

1. OS の起動には、管理者が認可した認証デバイスが必要となること
2. 管理者が、認証デバイスと利用する計算機の組み合わせに対し、起動可能とする OS を指定できること
3. 認証デバイスが抜き取られた場合、起動中の OS が停止すること

本研究で提案するシステムの概要を図 2.1 に示す。このシステムでは、管理者により認可された認証デバイスのことを単に**認証デバイス**と呼び、利用者が要求する OS を単に **OS** と呼ぶ。また、管理者により管理され、その OS が起動される計算機を単に**ハードウェア**と呼ぶ。このシステムは、認証デバイスとハードウェア、OS についての起動許可情報の管理を行う**認証サーバ**や、ディスクイメージの配布を行う**配布サーバ**と、利用者用の計算機である**クライアントシステム**から構成される。本研究のシステムでは、認証サーバは全てのクライアントシステムで 1 つとすることができ、かつ配布サーバは複数台のクライアントシステムごとに 1 つとすることができる。これにより、本研究のシステムの管理・運用上のコストを削減することが可能となる。

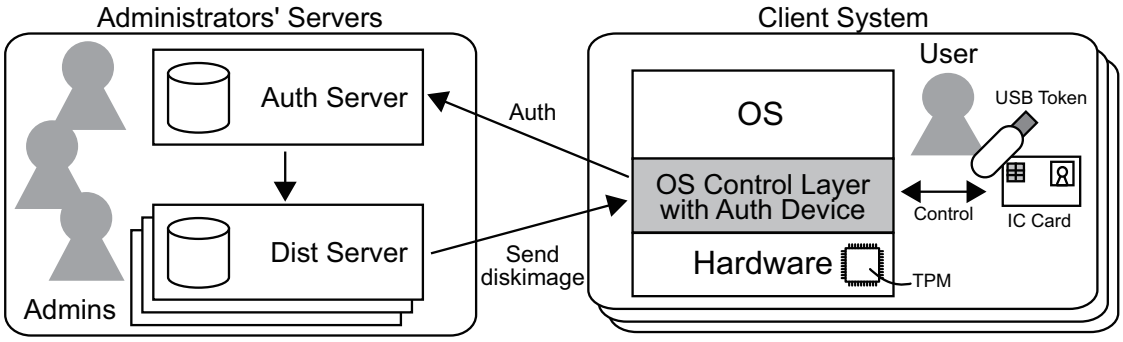


図 2.1: 起動・終了制御システムの概要

2.2.1 クライアントシステム

本研究で提案する手法では、クライアントシステムにおいて、OS 制御レイヤを用いて OS の起動・終了と認証デバイスへのアクセスを行う。これにより、OS 制御レイヤ上で動作する OS に変更を加えることなく、提案する認証・起動制御を実現することが可能となる。

本研究で提案する方式ではクライアントシステムの電源を投入しても認証デバイスが挿入されていない場合には OS は起動しない。OS 制御レイヤは認証デバイスが挿入されたことを検知すると、認証デバイスとハードウェア情報に応じて起動可能な OS の情報を取得し、利用者が選択した OS を起動する。また、物理的認証デバイスが抜かれたことを検知すると、起動した OS の停止処理を行う。これにより、本研究のシステムでは認証サーバに公開鍵情報が登録されている認証デバイスを用いて同時刻に起動可能な OS を高々 1 つとすることができる。

従来のコンピュータにおいて、ハードウェア上に搭載されている BIOS が OS の起動を行っている。したがって、この方式では OS 制御レイヤが BIOS に替わって起動・終了制御を行うものであり、従来のコンピュータシステムとの親和性が高い。BIOS は起動時のみ動作するのに対し、本制御レイヤは起動後も動作する。これにより、認証デバイスの抜き取りを監視することができる。また、ハードウェアや OS に対する変更点が少ないという利点がある。下位のレイヤから OS に対し状態変化を伝える例として、ACPI[28] がある。ACPI では、主に電源状態の変化を OS に伝える手段を提供しており、対応した OS であれば、その変化に応じて起動中の OS をシャットダウンさせることができる。本研究の手法においても、OS 制御レイヤが OS を停止させる方法として、この ACPI の状態変化を伝える方法を使うことができる。

2.2.2 認証サーバと配布サーバ

認証デバイスやハードウェアの情報、OS のディスクイメージなどを一括管理するために、本研究で提案する方式では認証サーバと配布サーバを設置する。認証サーバでは、次のような情報を管理、格納する。

(a) ポリシを記述した ACL(Access Control List)

(b) 認証デバイスに対応する公開鍵

(c) OS イメージ名と、その OS イメージを管理する配布サーバの URI

一方、配布サーバでは次のような情報を管理、格納する。

(d) OS イメージ名と OS イメージファイルの対応リスト

(e) OS イメージファイル

ハードウェアについては、SSL(Secure Socket Layer) クライアント認証により認証を行う。その正当性の検証は「クライアント証明書に署名を行った認証局が信頼できる認証局であるかどうか」について行われるため、配布サーバは認証局の証明書だけを持てばよい。

(a) では、利用を許可するハードウェア名、認証デバイス名、OS イメージ名の組み合わせを保持する。(b) では、認証デバイスに格納された証明書に対応する公開鍵情報を保持する。これにより、あらかじめ公開鍵情報を認証サーバに登録してある認証デバイスのみを本研究のシステムにおいて利用が許可された認証デバイスとすることができる。(c) では、OS イメージ名からその OS イメージを担当する配布サーバの URI を取得するために、その担当関係を保持する。最後に、(d), (e) は、クライアントに送信するディスクイメージの名前と実際のファイルとの対応を保持する。これにより、ディスクイメージファイルの管理が容易になる。

なお、本配布サーバでは、利用者に提供する OS 環境をディスクイメージとして管理する。これにより、種類の異なる OS を異なる OS 環境として利用者に提供することができるだけでなく、同一種類のオペレーティングシステムであっても、その上にインストールされているソフトウェアが異なれば、異なる OS 環境として利用者に提供することができる。

2.2.3 認証処理と OS イメージの転送

本研究の手法においては、起動可能な OS を決定する処理において、利用者と計算機の双方を同時に識別できることが求められる。利用者の識別には、利用者の持つ認証デバイスに格納された証明書 (利用者用証明書) を、計算機の識別には、ハードウェアモジュールに格納された証明書 (ハードウェア用証明書) を、それぞれ利用する。

ネットワークにおいて、安全に認証やデータ転送を行うプロトコルとして、HTTPS(Hyper Text Transfer Protocol Security) がある。HTTPS では、SSL サーバ認証とクライアント認証の両方を同時に用いることにより、サーバはクライアントを、クライアントはサーバを互いに識別することができる。このとき、本研究の方式ではサーバ側の証明書としては認証サーバまたは配布サーバを識別できる証明書を用い、クライアント側の証明書としてはクライアントシステムが実行されているハードウェアに関連づけられたハードウェア用証明書を用いる。これにより、通信経路が暗号化されるほか、認証サーバ側でクライアントシステムが実行されているハードウェアについての認証を行うことができる。

本研究の手法では、ハードウェアについての認証を行った上で、さらに利用者についての認証を行う必要がある。この認証処理は、「その利用者が、その計算機を用いて、認証処理を行っていること」を確実なものとする必要がある。つまり、認証処理を途中から別の計算機上で行うことにより、利用者と計算機の組み合わせを偽ることができないような認証の仕組みが必要となる。HTTPS クライアント認証を用いることにより、ハードウェア用証明書と利用者用証明書のうち、どちらか一つを識別することができるが、その双方を同時に識別することはできない。

そこで、本実装ではハードウェアについて認証を行ったセッション上で、HTTP とチャレンジアンドレスポンス方式を用いた独自プロトコルにより認証デバイスを用いた認証を行うこととした。提案するプロトコルにおける認証とイメージ転送の流れを次に示す。ここで、認証デバイス名とは、認証デバイスに格納された証明書の CN(Common Name) を

指す。この証明書は管理者により発行され、CN から利用者を一意に識別できるようなものである。また、その証明書の正当性は署名を検証することにより確認することができるため、認証デバイス名をもって、利用者を一意に識別することができる。

1. OS 制御レイヤは認証デバイスから認証デバイス名を取得する。
2. OS 制御レイヤは、認証サーバに対しハードウェア用証明書を用いた HTTPS クライアント認証セッションを確立する。このセッションを用いて、以下の処理を行う。
   (a) OS 制御レイヤは 1. で取得した認証デバイス名を名乗り、認証要求を出す。
   (b) 認証サーバはこのセッションを確立したハードウェア名 (ハードウェア用証明書の CN) を取得する。
   (c) 認証サーバは認証デバイス名とハードウェア名から、起動が許可されている OS とその OS イメージを担当する配布サーバの URI を決定する。
   (d) 認証サーバはランダムに作成したデータをチャレンジデータとして、起動が許可されている OS の情報とともに保存する。そのチャレンジデータと OS イメージを担当する配布サーバの情報を HTTP 応答としてクライアントに送信する。起動可能な OS が存在しない場合、そのハードウェアに対しその認証デバイスによる起動が許可されないことを表すため、HTTP エラー応答を送信する。
3. OS 制御レイヤは受け取ったデータから起動可能な OS のリストを取り出し、利用者に選択肢として提示する。
4. 利用者が起動する OS を選択すると、OS 制御レイヤは認証サーバから受け取ったチャレンジデータを、認証デバイスの署名機能を用いて署名し、レスポンスデータを作成する。
5. OS 制御レイヤは配布サーバに対し新規 HTTPS セッションを確立する。
   (a) OS 制御レイヤは 1. で取得した認証デバイス名とともに、レスポンスデータを配布サーバに送る。
   (b) 配布サーバは、受け取った認証デバイス名とレスポンスデータを認証サーバに送信する。
      i. 認証サーバは、受け取ったレスポンスデータをその認証デバイスに対応する公開鍵を用いて復号化する。
      ii. 復号化したデータと、認証要求時に保存しておいたデータを比較し、一致すれば保存しておいた OS 名を返す。一致しない場合はエラーを返す。
   (c) 配布サーバは、認証サーバから受け取った OS 名に対応する OS イメージを送信する。認証サーバからエラーを受け取った場合は HTTP エラー応答を送信する。

この認証プロトコルでは、通信に SSL を用いているため、鍵の機密情報が漏洩しない限り通信の安全性が保証される。また公開鍵暗号の理論から、認証サーバにおいてチャレンジデータとレスポンスデータの比較に成功した場合、その利用者がその計算機を利用していることが保証される。

2.3 起動・終了制御システムの実装

提案した方式の有用性を検証するために、クライアントシステムを仮想計算機モニタ Xen を用いて実装し、認証・配布サーバについては Web サーバ上の CGI プログラムとして実装した。

1 台のハードウェア上に複数の仮想計算機 (Virtual Machine:VM) を構築する仮想計算機技術の中で、Type-I VMM[26] と呼ばれるものは、BIOS と OS の間で仮想計算機モニタ (Virtual Machine Monitor:VMM) が動作する。VMM は実計算機資源の VM への割り当てや、VM の管理などを行うため、VM 上で動作するゲスト OS にとっては BIOS に相当するレイヤとみなすことができる。この特徴を利用し、今回は検証のために Type-I VMM を用いることとした。

本研究の手法で用いる VMM は、前述の Type-I VMM であればよい。今回は、オープンソースであり、ホスト OS として Linux を利用できるという理由から Xen を用いる。OS 制御レイヤをホスト OS の Linux を利用して実装することで、Linux が持つ HTTP や SSL などのライブラリ、ソフトウェアやデバイスドライバなどを利用して実現することができる。

Xen ではゲスト OS を DomainU と呼ばれる領域で起動し、DomainU は Domain0 と呼ばれる VMM 部分により管理される。利用者が要求した OS を DomainU として起動することにより、Domain0 で起動・終了制御を行うことができるようになる。この点に着目し、本研究では OS 制御レイヤを Xen の Domain0 上で実装することとした。ここでは、OS 制御レイヤを実装する Domain0 のことを**ホスト OS** と呼び、利用者が要求した OS のことを**利用者用 OS** と呼ぶ。

本実装では認証デバイスとして飛天ジャパン社の USB トークン ePass2000 を用いた。Domain0 の OS としては Linux 2.6.18(Fedora7) を、VMM としては Xen 3.2 を用いた。認証・配布サーバの OS としては Linux 2.6.18(Fedora7) を用い、HTTP デーモンとしては Apache 2.2.6 を、HTTP 上の認証プロトコル処理の実装には PHP 5.2.4 を用いた。また、認証サーバと配布サーバは同一ハードウェア上に実装した。

2.3.1 利用者用 OS の制御の流れ

本研究のシステムにおける利用者用 OS の起動の流れを図 2.2 に示す。電源が投入されると、本研究のシステムは、利用者に対してまず USB トークンの挿入を要求する。USB トークンが挿入されたことを検知すると、 本研究のシステムは USB トークンから取得した認証デバイス名と、あらかじめ Domain0 内に保存されているハードウェア用証明書を用いて、認証を伴う利用者用 OS イメージの取得処理を行う。

認証・配布サーバからの利用者用 OS イメージの取得が完了すると、そのイメージを仮想計算機上で起動し、利用者がその OS を利用できるようにする。また、同時にホスト OS 上で USB トークンの抜き取りを監視するデーモンを実行する。

利用者用 OS の稼動中に USB トークンが抜かれた場合、監視デーモンがそれを検知し VMM のシャットダウン命令を利用して利用者用 OS を停止させる (図 2.3)。

2.3.2 クライアントシステムの実装

本研究のシステムでは、クライアントシステムのハードウェアを識別するためにそれぞれのハードウェアに対してユニークな証明書をあらかじめ作成しておく。このハードウェア用証明書は、クライアントシステムに配置するホスト OS 内に同梱する。この証明書は、

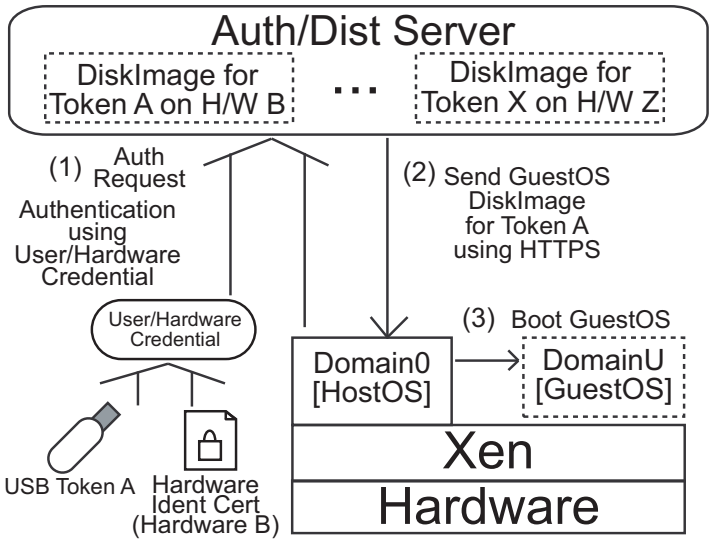


図 2.2: システム起動の流れ

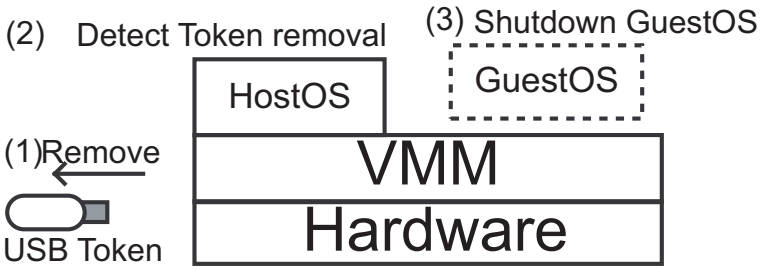


図 2.3: USB トークンが抜かれた場合

SSL クライアント認証で使用することができる X.509 証明書を用いた。X.509 証明書は広く利用されており、多数の証明書を容易に発行することが可能である。特に今回の目的においては、広く知られている CA による署名を受ける必要はないため、起動・終了制御システムの管理者が CA を設置し、その CA による署名を付与した証明書をハードウェア用証明書として利用することができる。ハードウェア用証明書の格納については、将来 TPM を用いることにより、より安全に実現可能となる。

本研究のシステムでは、利用者の識別には認証デバイスを用いる。この認証デバイスには、利用者に固有の証明書が格納されている。

認証デバイスがハードウェアから抜かれたときに、起動中の OS が停止されることを保証するために、ホスト OS では USB トークンの監視処理を行う。一度 USB トークンが抜かれたことが検出された場合、利用者用 OS はシャットダウンされる。この処理は次のような流れで動作する。まず、異なる USB トークンが挿された場合にそのことを検出できるように、利用者用 OS を起動する前に認証デバイス名[1]をホスト OS 上に記録しておく。

その後、ホスト OS 上で定期的に USB トークンにアクセスし、認証デバイス名が記録されているものと一致するかを確認する (ポーリング方式)。USB トークンにアクセスできない場合や、取得した認証デバイス名が記録されているものと異なる場合、起動に使用した USB トークンが抜かれたものとして判定する。

USB トークンが抜かれたことを検出した場合、まず DomainU をシャットダウンさせるコマンドを実行する。コマンドを実行すると DomainU にシャットダウン命令が送られ、このコマンドを受け取った DomainU はシャットダウンを開始する。

上記のコマンドを実行した後も定期的な確認を行い、DomainU のシャットダウンが確認できない場合には DomainU の強制終了コマンドを実行する。このコマンドは DomainU の状態に関係なく DomainU を強制終了させる。これは実際の計算機における電源断に相当する。

2.3.3 認証サーバと配布サーバの実装

2.2.3 項で示した方式を実現するために、認証サーバおよび配布サーバ上で動作する CGI プログラム、および認証サーバ・配布サーバ間の通信処理を PHP を用いて作成した。認証サーバ用 CGI プログラムは、クライアントシステムからの認証リクエストを処理し、配布サーバ用 CGI プログラムでは、クライアントシステムから送られたレスポンスデータの検証や、ディスクイメージの送信を行う。

認証リクエストの処理では、クライアントシステムから送出されるリクエストのパラメータとして認証デバイス名を受け取り、また環境変数からハードウェア用証明書の CN をハードウェア名として取得する。認証デバイス名とハードウェア名から起動すべき OS 名を ACL から検索する。また、OS 名とハードウェア名から、担当する配布サーバの URI を決定する。

クライアントに返すデータは、その OS を担当する配布サーバの URI とチャレンジデータを繋げたバイナリデータであり、それらは LF(LineFeed) を用いて区切ることとした。これにより、クライアントシステムは受け取ったデータを LF で分割し、前半を URI、後半をチャレンジデータとして扱うことが可能となる。なお、認証情報の偽造に対する強度が

[1]認証デバイスに格納された証明書の CN(Common Name) で、管理者は利用者を一意に識別することができるような CN を設定する。その正当性は、署名を検証することによって行うことができる

向上するため、チャレンジデータの長さは本研究において用いたUSBトークンにおいて署名を行うことができる最長のデータ長である48バイトとした[2]。

起動が許可される組み合わせであった場合、前述のURI・チャレンジデータをOS制御レイヤに返す。また、配布サーバからの問い合わせのために、OS制御レイヤが起動しようとしているOSの名前と、送信したチャレンジデータをサーバ上に保存しておく。

もしACLからの検索結果に対応するOSがない場合には、その認証デバイスとハードウェアの組み合わせではOSの起動が許可されていないことを表すため、この処理のHTTP応答としてステータスコード404 NotFoundを返す。

認証サーバ・配布サーバ間の通信処理では、認証サーバは配布サーバから送出されるリクエストのパラメータとして、認証デバイス名とレスポンスデータを受け取る。認証サーバはそのパラメータをもとに、その認証デバイスに対応する公開鍵を検索する。この公開鍵を用いてレスポンスデータの復号化処理を行い、保存してあるチャレンジデータとの比較を行う。比較の結果によらず、復号化処理を行った場合には一時的に保管したチャレンジデータは削除する。比較処理の結果、一致した場合には、この処理の応答として、一緒に保存していたOS名を返す。一致しない場合は、HTTP応答としてステータスコード404 NotFoundを返す。

レスポンス処理では、クライアントシステムから送出されたリクエストのパラメータとして認証デバイス名とレスポンスデータを受け取る。受け取ったパラメータを用い、認証サーバ・配布サーバ間の通信処理を実行する。正常なHTTP応答が返ってきた場合には、そのボディからOSイメージ名を取得し、クライアントに対する応答として対応するイメージをボディとして送信する。エラー応答が返ってきた場合には、クライアントに対しHTTP応答としてステータスコード404 NotFoundを返す。

2.4 評価

本研究で提案する方式の有用性の評価を行うため、異なる証明書が格納された3つのUSBトークンと、2つのハードウェアを用意し、実装されたシステムを用いて利用者用OSの起動制御実験を行った。また、本システム特有の処理にかかる時間の計測を行った。

2.4.1 ハードウェア構成

実験には、次のようなハードウェア構成のサーバ・クライアントを用いた。それぞれのハードウェアは、図2.4に示すように、1000BASE-TのEthernetで接続されており、USBトークンとしてToken A, B, Cの3つと、クライアントシステムとしてHardware α, βの2台から構成される。

- クライアント Hardware α
  - CPU: Intel PentiumD 820 2.8GHz
  - Memory: DDR2 SDRAM 1024MB
  - HDD: SerialATA 160GB
- 認証・配布サーバ / クライアント Hardware β

[2]使用した認証デバイスePass2000のデバイスドライバが未完成であり、一度に署名可能なデータが48バイトまでに制限されていたため。

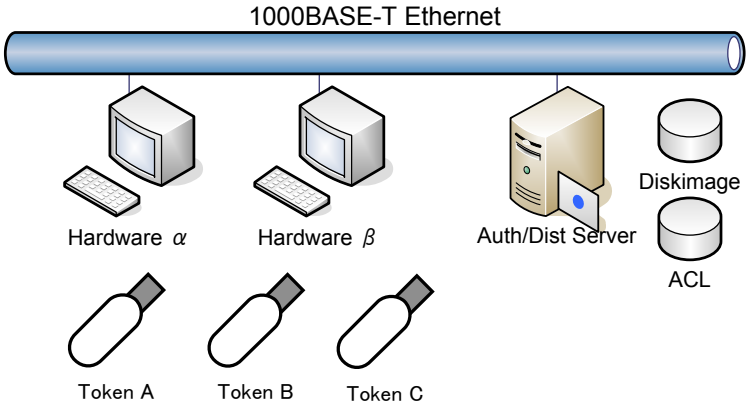


図 2.4: 実験環境概略図

- CPU: Intel Core2Quad Q6600 2.4GHz
- Memory: DDR2 SDRAM 4096MB
- HDD: SerialATA 250GB

2.4.2 実験方法

まず初めに、表 2.1 に示すような起動許可設定を行った。また、異なる証明書が格納された USB トークン Token A, B, C を用意し、対応する公開鍵をそれぞれ認証サーバに格納した。

次に、Hardware α 上にてクライアントシステムを稼動させ、Token A, B, C をそれぞれ挿入し、どの利用者用 OS が起動するかを観察した。また、その OS の起動処理に掛かる時間を計測した。最後に、Hardware β 上にてクライアントシステムを稼動させ、Token A, B, C をそれぞれ挿入し、どの利用者用 OS が起動するかを観察した。

利用者から見た起動所要時間とは、USB トークンを挿入してから、利用者が利用者用 OS にアクセスできるようになるまでの時間である。その時間は、認証にかかる時間、OS イメージの転送にかかる時間、利用者用 OS の起動時間に分けることができる。しかし、利用者用 OS の種類や構成によりその起動時間は様々である。そこで、本研究のシステムの評価として、認証にかかる時間を計測した。これは OS 制御レイヤに処理が移ってから、OS イメージの受信が始まるまでの時間で、ここでは**認証フェーズ所要時間**と呼ぶ。この所要時間が OS イメージの転送にかかる時間に比べどの程度かかるかを比較するため、OS イメージの受信開始から受信完了までの時間の計測も行った。ここではこれを**転送フェーズ所要時間**と呼ぶ。所要時間の測定は、それぞれの USB トークンについて 3 回ずつ行った。なお、OS A, B ,C の OS イメージサイズは、それぞれ 540MB, 1080MB, 1540MB とした。

2.4.3 実験結果

Hardware α を用いた起動実験では、Token A の挿入時には OS A が、Token B の挿入時には OS B が、Token C の挿入時には OS C が、それぞれ起動した。Hardware β を用いた起動実験では、Token A を挿入したときは OS A が、Token B の挿入時には OS C が起

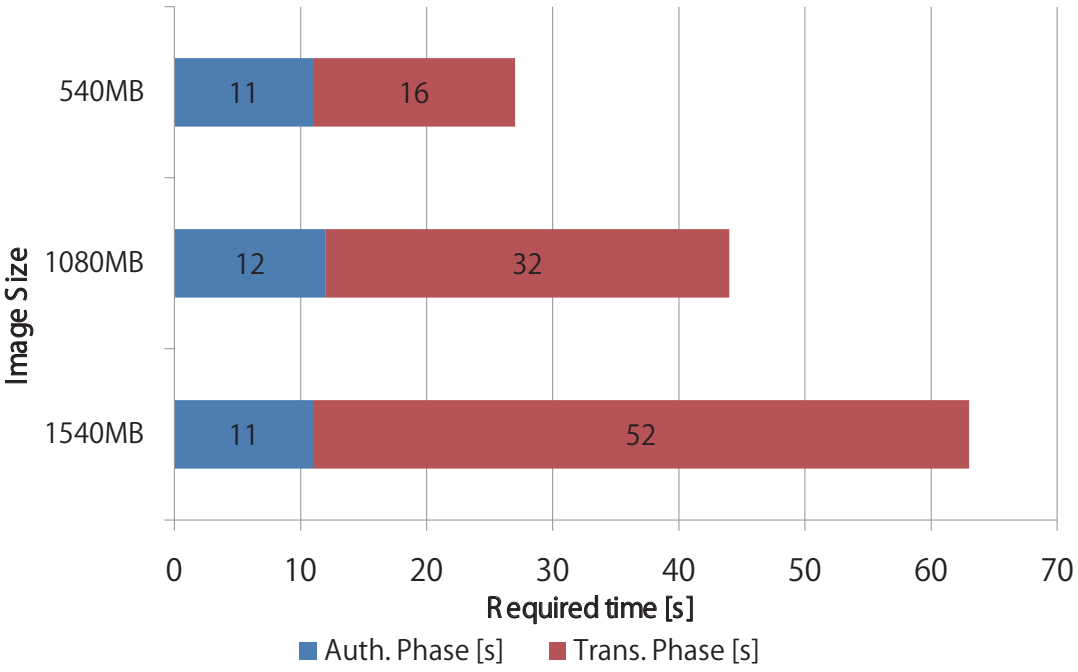


図 2.5: 平均所要時間計測結果

動したが、Token C を挿入した時には OS は起動しなかった。これにより、表 2.1 の設定どおりに起動制御が行われていることが確認できた。

また、Hardware α を用いた起動実験において、ディスクイメージのサイズが 540MB, 1080MB, 1540MB である 3 種類の OS(OS A, B, C) について、認証フェーズと転送フェーズの所要時間をそれぞれ 3 回計測した。その実験結果を表にまとめたものを表 2.2 に、それをグラフに表したものを図 2.5 に示す。結果より、すべてのイメージサイズで認証フェーズの所要時間は約 11 秒と一定であるが、転送フェーズの所要時間はディスクイメージサイズにほぼ比例することが読み取れる。

表 2.1 の設定では、Token A に着目すると、全てのハードウェアで OS A が起動した。一方、OS A に着目すると、OS A の起動を許可されているトークンは Token A だけであるため、OS A の最大稼動数は 1 となる。これにより、最大稼動数を管理されている OS がハードウェアの制限なくどこでも稼動可能となっていることがわかる。

それに対し、Token B では、Hardware α では OS B が起動し、Hardware β では OS C が起動した。これにより、認証デバイスに対して、異なるハードウェアでは異なる OS が起動するような柔軟な起動制御が実現されていることが確認できた。Token C と OS C に着目すると、Hardware α でのみ OS が起動した。これにより、「起動を許可しない」という設定も可能であることが確認できた。以上より、柔軟な OS の起動制御が実現できてい

表 2.1: OS 起動許可設定

	Hardware α	Hardware β
Token A	OS A	OS A
Token B	OS B	OS C
Token C	OS C	No OS permitted

るといえる。

また、所要時間の計測結果では、認証フェーズにおける平均所要時間は約 11 秒であった。今回使用した USB トークンでは、認証デバイス名の取得に約 4 秒かかり、チャレンジデータの署名に約 7 秒かかった。両者を合わせると約 11 秒となり、認証フェーズにおける平均所要時間の大半が USB トークンアクセスにかかる時間である。USB トークンに対する処理以外に要した時間は 1 秒以下であり、無視できる程度であった。

本研究のシステムでは、配布サーバがどのトークンとハードウェアの組み合わせを担当するかという情報を認証サーバが一元管理しているため、クライアントシステムの数が増加した場合にも、配布サーバを追加することで対応することができる。

このとき、1 台の配布サーバがどの程度の数のクライアントシステムを処理できるかが重要になる。配布サーバの性能により、CPU の SSL 処理に要する処理能力または回線の帯域幅によるボトルネックが生じる。そのため、それぞれの利用者用 OS イメージの転送に要する時間の許容量から、配布サーバの台数や性能を見積もった上で設置する必要がある。この問題に関しては、次章で詳しく述べる。

2.5 まとめ

本章では、集中管理された利用者用計算機のうち、大学の計算機実習室に配置された計算機において利用者認証を強化することにより利便性を向上させる手法について述べた。これを実現するために、利用者と計算機の双方を識別し、その組み合わせに応じて OS の起動・終了制御を行う仕組みを提案した。この仕組みを用いることで、従来はそのための CD-ROM を作成していたようなより高い権限を使いたい場合においても、集中管理された計算機により対応することを可能とした。管理者が利用者と計算機の組み合わせに応じて起動可能な OS を設定することができるため、管理者が認めた場合、スキルの高い利用者に対して、その利用者が必要とするような OS を起動することを許可することができる。これによって、集中管理された計算機を用いて、利用者を限定することにより高い権限を与えることを可能とし、利便性を向上させることを可能とした。

この利用者と計算機の組み合わせに応じた OS の起動・終了制御の仕組みを実現する上での技術的課題として、利用者を識別するための認証デバイスと、計算機を識別するための TPM のようなハードウェアモジュールの双方を同時に識別し、認証を行う仕組みの実現がある。従来の証明書を用いた認証では、高々 1 つの証明書を用いた認証しか行うことができないため、2 つの証明書を同時に用いて認証を行うことができず、利用者と計算機の双方を同時に識別することができない。そこで本研究では、計算機を識別するための証明書を用いて HTTPS クライアント認証セッションを確立し、その上でさらに、利用者を識別するための証明書を用いてチャレンジアンドレスポンス方式の認証を行う仕組みを提

表 2.2: 所要時間計測結果

Image	Auth Phase[s]				Trans Phase[s]			
Size[MB]	1st	2nd	3rd	Avg	1st	2nd	3rd	Avg
540	11	11	11	11	15	16	17	16
1080	13	11	11	12	31	32	33	32
1540	11	10	11	11	52	52	53	52

案、実装した。これにより、その利用者がその計算機を用いて認証を行っていることを確実に識別可能とした。

このような起動・終了制御システムを実装し、その有用性を評価した。その結果、利用者と計算機の組み合わせに応じた OS の起動・終了制御を実現できることを示した。また、OS イメージの配信について、そのイメージサイズが起動処理にかかる時間に影響することを示し、配信を行うサーバがボトルネックとなることについて述べた。この問題とその解決方法に関しては、次章で詳しく述べる。

第3章　ネットワークブートシステムにおける P2P 方式ディスクイメージ配信

前章では、利用者と計算機の組み合わせに応じて OS の起動・終了制御を行うシステムについて述べた。このシステムでは、起動が許可された OS のディスクイメージをネットワーク経由で取得し、起動していた。このシステムに限らず、従来のネットワークブートシステムでは、この OS のディスクイメージの配信がボトルネックとなる。本章では、このネットワークブートシステムにおける OS ディスクイメージ配信のボトルネックを解消するための P2P 方式のディスクイメージ配信システムについて述べる。

ネットワークブートシステムにおける OS イメージ配信のボトルネックは、大学の実習室のような環境において時として問題を引き起こす。実習室においては、授業の開始時や教員の指示などにより、多数の計算機がほぼ同時に起動処理を開始するという状態がしばしば発生する。そのような状態が起こると、従来のネットワークブートシステムでは配信サーバがボトルネックとなり、起動に長い時間がかかってしまうという問題が生じる。これは授業の進行の妨げとなってしまうため、大学はこのボトルネックが生じないよう、設備に対して投資を行っている。

具体的には、筑波大学では 985 台の計算機をキャンパス内に設置しており、これらの計算機に対してネットワークブートシステムを構築している。配信サーバは 20 台用意され、末端の計算機までのネットワークは、教室に設置された計算機の台数に応じて、最大 10Gbps という高速なものを導入している。このように、高速なネットワークや多数の配信サーバを用意することにより、起動処理の集中による起動所要時間の悪化を防いでいる。

このボトルネックを解消するために、本研究では P2P 方式のディスクイメージ配信手法を提案する。実習室のような環境では、同じ教室、同じ時間であれば同じ OS が利用される傾向がある。これはつまり、同じ OS のディスクイメージが要求されることを意味する。もしネットワークブートシステムのクライアントが、配信サーバからだけでなく、同じ OS を起動している他のクライアントからもディスクイメージを取得することができれば、配信サーバの負荷は軽減され、そのトラフィックを削減することができる。これにより、大学は配信サーバやネットワークに対し高価な支出を行わなければならない事態を回避することができる。

ネットワークブートシステムにおいて、その配信におけるボトルネックを軽減する手法は、今までにいくつか提案されている。須崎らは、ネットワーク経由で KNOPPIX を配信する HTTP-FUSE KNOPPIX[56] を提案している。この HTTP-FUSE KNOPPIX では、インターネット経由での OS 配信を目的としているため、ファイアウォールを越える問題を解決するために、ディスクイメージの配信に HTTP を用いる。これにより、クライアントにおいてそのディスクイメージを受信する際、CDN(Contents Derivery Network) を用いることが可能となり、容易に配信システムの負荷を分散することを可能としている。また、金井ら [66] は、この HTTP-FUSE KNOPPIX を用いて P2P 方式の配信を行うことにより、配信システムの負荷を分散し、KNOPPIX を用いたシンクライアント OS を効率良く配信する手法を提案している。この手法では、配布サーバがクライアントの起動段階

に応じてツリーを構成し、そのツリーに応じて上位のノードを決定し、そのノードから必要なブロックを取得する。このどちらの手法も、Linux のディストリビューションである KNOPPIX を配信対象としている。

これらに対し、本研究は、大学における実習室のような環境を対象とする。そのため、高速かつ低遅延なネットワークで接続された多数の計算機に対し、Microsoft Windows のようなプロプライエタリな OS を含めて多数の OS を配信可能であることが求められる。また、そのような環境では、授業時間や教員による指示などに基づき多数の計算機が起動処理を開始するという状況がしばしば起こる。このような起動処理の集中はネットワークブートシステムの性能に大きく影響を与えてしまう。金井らの研究 [66] においても、ワーストケースである一斉に起動処理を開始する場合について、その性能が最悪のものであることを示している。しかし、実習室のような環境で運用を行う場合、完全に同時に起動処理が開始されることはまれであり、実際には正規分布に従うような分布で起動処理が行われる。そこで本研究では、現実的な起動処理の分布として、正規分布に従って多数の計算機が起動処理を開始するような場合に、その起動所要時間がどのように変化するかについて明らかにする。

本研究で提案する P2P 方式のディスクイメージ配信手法は、**配信サーバ**と**クライアントノード**から構成される。クライアントノードは、ディスクイメージのブロックを配信サーバからだけでなく、同じ OS を起動している近くのクライアントノードからも取得する。配信サーバは、BitTorrent におけるトラッカとしての役割も担い、新たに起動するクライアントノードに対し、ブロックを提供することができるノードの情報を提供する。クライアントノードは、OS 制御レイヤと同様に仮想計算機を用い、Microsoft Windows を含む様々な OS を、OS に対する変更なしに動作させることができる。この OS はゲスト OS としてクライアントノード上で動作し、そのディスクアクセスは P2P 処理を行うプログラムによりフックされる。このプログラムは、配信サーバや他のノードと通信し、必要となったブロックを取得したり、ブロックやそのメタ情報を交換したりする。

このような配信手法を実装し、その有用性を確認するために評価実験を行った。実験は筑波大学の実習室で行われ、112 台のクライアントノードを用いて、その起動タイミングを変化させながら起動所要時間を測定した。その結果、従来の NFS による手法に比べ、本研究の手法では起動処理が集中したような場合においても起動にかかる時間を短縮することができることを示した。本研究の手法では、クライアント間でデータのやりとりを行うことにより、配信サーバのトラフィックを削減することができる。それだけでなく、正規分布に従うような起動分布においても、先に起動しているクライアントからディスクイメージのデータを取得することができることから、全てのクライアントが起動を完了するまでにかかる時間を短縮することができることを示した。これらの結果から、配信サーバからクライアントまでのネットワークを広帯域の回線で結ぶことなく、配信サーバのボトルネックを解消できることを示した。

3.1 関連研究

本節では、ネットワークブートシステムにおけるディスクイメージ配信システムに関連する研究について述べる。

3.1.1 P2P ファイル共有システム

大容量のデータを効率良く多数のコンピュータに提供するための技術として、P2P ファイル共有システムがある。これは、あるコンピュータから配信されるデータを、受信したいコンピュータ同士で融通することにより効率的に多数のコンピュータに配信するシステムである。Freenet[13] や Napster[53]、Gnutella[49] や Kazaa[40]、BitTorrent[14] などはその一例である。

これらの P2P ファイル共有システムは、大容量のデータを効率良く配信することを目的としている。そのため、配信対象のファイルを構成するブロックを、提供可能なノードからかき集める処理を行う。ネットワークにどのノードが参加しているかという情報や、どのノードがどのブロックを持っているかという情報は、トラッカ (Tracker) と呼ばれるサーバが管理している。

本研究の手法では、ネットワークブートに使用する OS を格納したディスクイメージを効率良く配信することを目的とする。OS のディスクイメージもまた大容量のデータであるが、必ずしもその全てのデータが必要であるとは限らないという特徴がある。たとえば、あるアプリケーションのデータを格納している領域は、そのアプリケーションを起動しようとしない限り不要であることが多い。加えて、その OS を起動しようとした場合、OS の起動処理で必要となるファイルを格納した領域がダウンロードできない場合、起動処理を続行することができずブロックしてしまい、利用者はその間待たされてしまう。そのため、ネットワークブートに用いる OS のディスクイメージを配信する場合、あるブロックを要求された場合に、速やかにそのデータを利用可能とすることができる仕組み、すなわちオンデマンドアクセスの仕組みが必要不可欠である。本研究の手法では、このオンデマンドアクセスの仕組みを提供するほか、配信サーバの負荷を軽減するため、ネットワーク的に近いノードと積極的にデータを交換する仕組みを提供する。

O'Donnell[47] は、BitTorrent を用いて、大学の実習室で用いる OS を格納したディスクイメージを配信する手法を提案している。この手法では、ディスクイメージのブロックは夜間や休日のようなアイドルタイムに配信され、実習室の計算機上に保存される。利用者が計算機を利用するときには、この配信されたディスクイメージを用いて仮想計算機として起動する。従来の Norton Ghost[58] を用いた配信方法では、ある計算機が受信したデータが破損していた場合、そのときに行われている転送処理が全て終わってから、改めて破損していた部分を送信し直す必要があり、配信効率が悪いという問題があった。BitTorrent を用いることにより、この再送処理を計算機ごとに、必要なブロック単位で行うことを可能としている。

本研究の手法では、同様に大学の実習室で用いる OS を配信することを目的としている。O'Donnell の方法では、アイドルタイムに各計算機にディスクイメージを配信するため、配信完了後に計算機を利用する場合、ディスクイメージのデータは計算機に接続されたストレージから読み込まれる。そのため、利用の際にネットワーク通信を必要とせず、そのアクセス速度は普通に OS をインストールした場合とほぼ同等の速度となる。本研究では、このような事前に配信を行うような方法ではなく、OS のディスクイメージを必要になった段階でネットワーク経由で転送するような方法を対象に、P2P 方式でボトルネックの解消を図る。

Spotify[38] は、P2P 方式の音楽配信サービスである。Spotify では、配信対象とするデータを音楽データに限定し、その特徴に基づいて最適化を行っている。利用者の計算機上には、数 GB 程度の、比較的大容量のキャッシュ領域が設けられる。利用者が音楽を再生する場合、同じ楽曲を複数回聴くことが多いという特徴からキャッシュのヒット率は高く、

P2P ネットワークの負荷軽減に効果がある。また、楽曲を再生している間に、プレイリストにおける次の楽曲のデータを先読みしたとしても、その先読みが無駄になる確率が低いため、先読みの効率がよいという特徴がある。もし再生すべき楽曲がキャッシュ上に存在しない場合には、先頭 15 秒のデータを Spotify 社の配信サーバから取得し、その 15 秒のデータを再生している間に、残りのデータを持つノードを探し、ダウンロードを行う。もし残りのデータを持つノードが見つからなかった場合には、同じく Spotify 社の配信サーバからダウンロードする。このような仕組みにすることで、Spotify 社の配信サーバの負荷を軽減している。

本研究の手法も同様に、オンデマンドアクセス機能を提供し、また配信データの特徴に基づいて最適化を行っている。本研究の手法では、ネットワークブートに用いる OS を格納したディスクイメージを配信対象としている。このデータには、次のような局所性を持つという特徴がある。大学の実習室のような環境においては、授業で計算機を使う場合のように、同じ部屋で、同じ時間であれば、同じ OS が利用されることが多い。その場合、同じ OS を利用するだけでなく、同じアプリケーションが利用される可能性が高い。つまり、同じディスクイメージの、同じ部分が必要とされる可能性が高いといえる。さらに、同じ部屋に設置された計算機は、通常同じネットワークスイッチに接続されていることが多い。これらの特徴から、ネットワーク的に近い計算機同士で、ディスクイメージのブロックを交換することにより、配信サーバの負荷を軽減し、効率的な配信を実現可能としている。

3.1.2 分散ファイルシステム

Ceph[63] は、3 種類のサーバを組み合わせて用いることにより、大容量かつ高信頼性を実現した分散ファイルシステムである。Ceph では、システム全体を管理するモニタ、メタデータを管理するメタデータサーバ、オブジェクトのデータを格納するオブジェクトストレージデバイスの 3 種類のサーバから構成される。これらのサーバを複数台組み合わせて多重化することにより、大容量・高信頼性を実現している。RADOS (Reliable, autonomic distributed object store) は、この Ceph のバックエンドで利用されている分散オブジェクトストアシステムである。仮想計算機モニタ KVM では、RBD (RADOS block device)[3] を用いることにより、ディスクイメージの格納に RADOS を利用することができる。

Sheepdog[43] は、仮想計算機のゲスト OS を格納したディスクイメージを、P2P ネットワーク上に格納するシステムである。Sheepdog では、多数の Sheepdog ノードから構成される P2P ネットワーク上に、ディスクイメージを配置する。その際、複数のノード上にレプリカを置くことにより、高信頼性を実現している。

これらの手法は、多数のサーバを用いて、大容量かつ信頼性の高いストレージを実現し、そのストレージ上のデータをクライアントに提供する。本研究の手法では、配信効率を向上させるため、配信サーバからクライアントへの転送だけでなく、クライアント同士でデータを交換する。

3.1.3 インターネット経由による KNOPPIX 配信

インターネット上に設置された配信サーバを用いて OS を配信することにより、計算機への OS のインストール作業の必要なしに、様々な OS を利用可能とするための手法が複数提案されている。

Suzaki らは、そのような OS 配信技術として、まず SFS-KNOPPIX[55] を開発した。これは、様々な PC 上で動作する Linux である KNOPPIX を、SFS (Self Certifying File System)

を用いて配信する技術である。この SFS-KNOPPIX を用いて、Suzaki らは、レイテンシが低い環境であれば、KNOPPIX を CD から起動する場合よりも早く起動を完了することを示した。また、レイテンシが MAN (Metroporitan Area Network) 規模以上の場合、CD を使用する場合よりも起動にかかる時間が悪化することを示した。

その後、SFS が持つ拡張性の問題である、ファイアウォール越えの問題や、レイテンシの問題などを解決するため、HTTP-FUSE KNOPPIX[56] を開発した。HTTP-FUSE KNOPPIX では、その名の通りデータの転送に HTTP を用いるため、一般的な環境において、容易にファイアウォールを越えて通信することが可能である。また、レイテンシの問題を解決するため、DLAHEAD と呼ばれる起動に必要なブロックを前もって取得しておく機能を持つ。これにより、ネットワークのレイテンシの影響を軽減している。

本研究の手法では、これらと同様に、計算機へのインストール作業なしに様々な OS を提供することを目指す。HTTP-FUSE KNOPPIX では、インターネット上に配信サーバを設置し、様々な場所から OS をブートすることを目指している。そのため、通信がファイアウォールによる影響を受けないことが望ましいことから、配信に用いるプロトコルは一般的に利用されている HTTP を用いている。また、配信側の負荷を軽減・分散するために、HTTP で広く利用される負荷分散技術である CDN (Contents Derivery Network) を用いている。一方、本研究が対象とする環境は、低レイテンシのネットワークで接続された大学の実習室のような環境である。このような環境においては、その低レイテンシという利点を活かせるよう、通信におけるボトルネックを極力軽減することが望ましい。そのため本研究の手法では、独自プロトコルを実装している。

3.1.4 P2P モバイルシンクライアントシステム

金井ら [66] は、HTTP-FUSE KNOPPIX を用いて P2P 方式でネットワークブートを行うシンクライアントシステムを提案している。この方法では、配布サーバはクライアントに対し、シンクライアント OS として KNOPPIX を配信する。クライアントはネットワークブートを行う際、配信サーバからだけでなく、他のクライアントからも必要なブロックを取得することができる。

配信サーバは、クライアントにブロックを配信するだけでなく、現在起動しているクライアントについて、それぞれのクライアントがどの起動段階にいるかを管理している。これは、配信対象の KNOPPIX について、その起動処理に手を加え、配信サーバにハートビートを送る際に自分が今どの起動段階にいるかを報告させることにより実現している。配信サーバは、各クライアントの起動段階に応じて配信ツリーを構築する。クライアントはこのツリーの情報を用いて、上位のノードからブロックを取得することができる。

クライアント側では、HTTP によるブロックの取得・配信のために Web プロクシソフトウェアである Squid[4] を稼動させる。動作に必要なブロックを取得する際には HTTP を用いるが、その URL へリクエストを送る際、まずホスト名の解決のために配信サーバへ DNS 問い合わせを行う。配信サーバはこの問い合わせに対して、配信ツリーに応じて、他のクライアントの IP アドレスを返す。これにより、クライアントは自らの上位に位置するクライアントに対してリクエストを送ることができる。リクエストにより得られたブロックデータは Squid によりキャッシュされ、他のクライアントに提供される。

本研究の手法も同様に、ネットワークブートの際に必要となるブロックデータを、クライアント間で転送することによりトラフィックを削減することを目的とする。金井らの手法では KNOPPIX を配信対象とするが、本研究では、大学の実習室のような環境を対象とし

ているため、様々な種類の OS を配信できることが必要である。特に、Microsoft Windows のような、プロプライエタリな OS を配信できることが必要となる。

そのため、対象となる OS の起動処理に手を加えることにより起動段階を配信サーバへ伝達し、その情報を用いて配信処理を効率化することは容易ではない。起動開始時から取得したブロックの数を起動処理の段階を決定するために使えば、ある程度の正確さで起動段階を推定することは可能であると考えられるが、その推定処理は配信対象の OS に依存してしまう。

金井らの研究では、8 台のクライアントを同時に起動した場合、10 秒ごとに 1 台ずつ起動した場合、30 秒ごとに 1 台ずつ起動した場合についての評価を行っている。その結果として、いずれの場合においても、P2P 方式を用いることにより、用いない場合に比べ起動時間を高速化できたことを示している。

本研究で対象とする実習室のような環境においては、授業時間や教員による指示などに基づき、多数の計算機が起動処理を開始するという状況がしばしば起こる。このような起動処理の集中はネットワークブートシステムの性能に大きく影響を与え、特に授業に利用するような場合には、授業の進行を妨げてしまう可能性があるため、大きな問題となる。ネットワークブートシステムでは、起動を行おうとしている全てのクライアントが一斉に起動を開始する場合がワーストケースとなる。金井らの研究においても、同時起動の場合の結果が、時間を空けて 1 台ずつ起動した場合よりも悪化している。しかし、実際の実習室における運用において、秒単位で一斉に起動処理を開始するようなことは見られない。これは、起動開始操作が、授業の開始や教員による指示などを受けて、人間の手によって行われるためである。実際の実習室における起動処理の分布を見ると、正規分布に従うことがわかる。そこで本研究では、ワーストケースである一斉に起動処理を開始するような場合だけでなく、正規分布に従い多数の計算機が起動処理を開始するような場合に、その起動所要時間がどのように変化するかについて明らかにする。

金井らの方法では、彼らが既に提案している耐障害性向上手法 [35] を適用可能とするため、ツリーの構成と他ノードの情報の解決に DNS-Balance 改を用いている。また、HTTP-FUSE-KNOPPIX と同様に、ファイアウォールを越えて配信を行うため、ブロックの転送に HTTP を用いている。本研究の手法では、キャンパスネットワーク内ネットワークブートシステムにおける配信効率を向上させることを目指している。そのため、ファイアウォールを超えられるというメリットよりも、オーバヘッドが小さく、より低レイテンシで通信できることを重要視しているため、独自プロトコルによるブロック・メタ情報交換を行っている。

3.1.5 BitVisor を用いたゲスト OS に透過的なネットワークブートシステム

表ら [68] は、BitVisor を用いて、ゲスト OS に透過的なネットワークブートの仕組みを実現している。この方法では、BitVisor はゲスト OS に対して ATA デバイスが存在しているように振る舞う。ゲスト OS はこの ATA デバイスに対して操作を行うが、その操作は BitVisor によりフックされ、AoE (ATA over Ethernet) プロトコルに変換され、実際にディスクイメージを提供する AoE サーバへ転送される。このような方法を取ることにより、ATA デバイスを操作することができるゲスト OS であれば、透過的にネットワークブートを実現することが可能となる。

本研究の手法も同様に、ゲスト OS の何らかの動作に応じて、仮想計算機モニタのレイヤで何らかの処理を行う仕組みを持つ。表らの方法では、BitVisor を用いてゲスト OS に

透過的なネットワークブートシステムを実現している。BitVisor は準パススルー型の仮想計算機モニタであるため、デバイスのエミュレーション機能を持たない。そのため、表らはゲスト OS にディスクデバイスを提供するため、ATA/AoE を用いている。

本研究の手法では、ネットワークブートにおける配信システムのボトルネックを解消することを目的として、仮想計算機モニタのレイヤにおいて P2P 方式のディスクブロック交換機能を実現している。また、既存のライブラリやプログラミング環境を利用することができるという理由から、ホスト OS として Linux を、仮想計算機モニタとして KVM を用いている。KVM を用いた仮想計算機構築環境では、ゲスト OS に提供するディスクデバイスとして、ディスクイメージファイルを利用することができる。加えて、FUSE (Filesystem in Userspace) を使うことにより、ファイルアクセスをフックするプログラムを容易に記述することができる。本研究では、これらを用いて P2P 方式によりゲスト OS にディスクデバイスを提供している。

3.2 従来のネットワークブートシステムの問題点と P2P 方式配信システム

従来の NFS-root Linux[34]、Citrix 社の Citrix Provisioning Services (PVS)[12]、Apple 社の NetBoot[6] などに代表されるネットワークブートシステムでは、高い処理能力を持つストレージサーバと高速なネットワークを必要とする。これは、前章で述べた認証デバイスを用いた OS の起動・終了制御システムの実装においても同様である。本研究では、そのようなストレージサーバやネットワークなしに、効率良く OS の配信を行うことができるネットワークブートシステムを提案する。このシステムは、従来のネットワークブートシステムに比べ、以下のような機能を備える点で異なる。

P2P ディスクブロック共有機能
: クライアントノードはディスクブロックを配信サーバからだけでなく、他のクライアントノードからも取得することができる

負荷分散
: 複数のクライアントノードがブロックを提供可能である場合、どのノードにも負荷が分散される

図 3.1 に、本研究で提案する P2P 方式のネットワークブートシステムの概要を示す。このシステムは、中央サーバとクライアントコンピュータ群、そしてそれらを接続するネットワークから構成される。中央サーバは、Citrix の PVS や Apple の NetBoot と同様に、ディスクイメージを格納し、クライアントに配信する。それに加え、中央サーバは BitTorrent におけるトラッカのようにも振る舞い、負荷分散のための情報を提供する。この中央サーバは、中央管理を行うサーバルームに設置される。

クライアントコンピュータ群は教室などに設置され、利用者に提供される。クライアントコンピュータの概要を図 3.2 に示す。図に示されるように、クライアントコンピュータ上では 2 つの OS が実行される。1 つ目は、ホスト OS である。ホスト OS では、仮想計算機モニタ (VMM) と、本研究で提案する P2P 方式の処理を行うプログラムである **P2P ディスクイメージプロバイダ**を実行する。2 つ目は、ゲスト OS である。このゲスト OS は、ホスト OS 上の仮想計算機として実行され、利用者に提供される。本研究ではこのゲスト OS

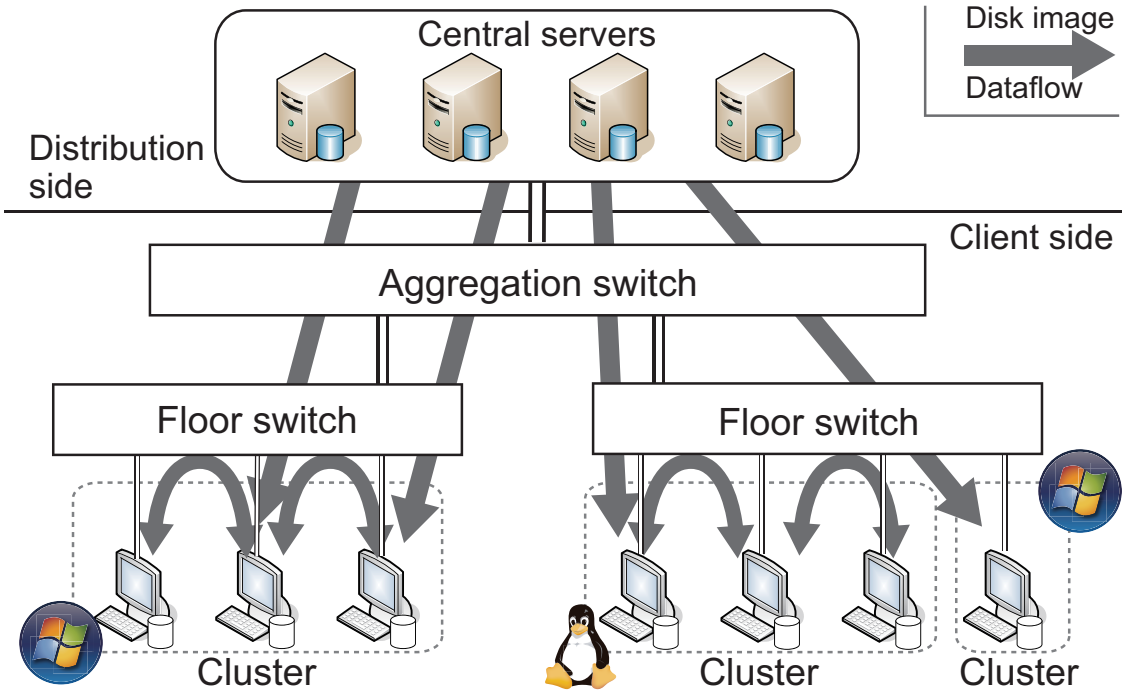


図 3.1: P2P 方式によるディスクイメージ配信システムの概要

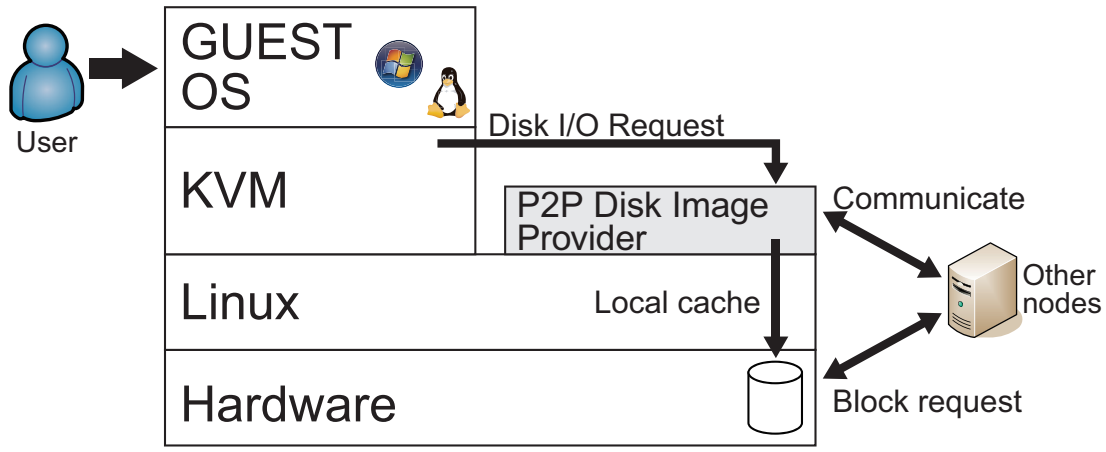


図 3.2: クライアントシステムの概要

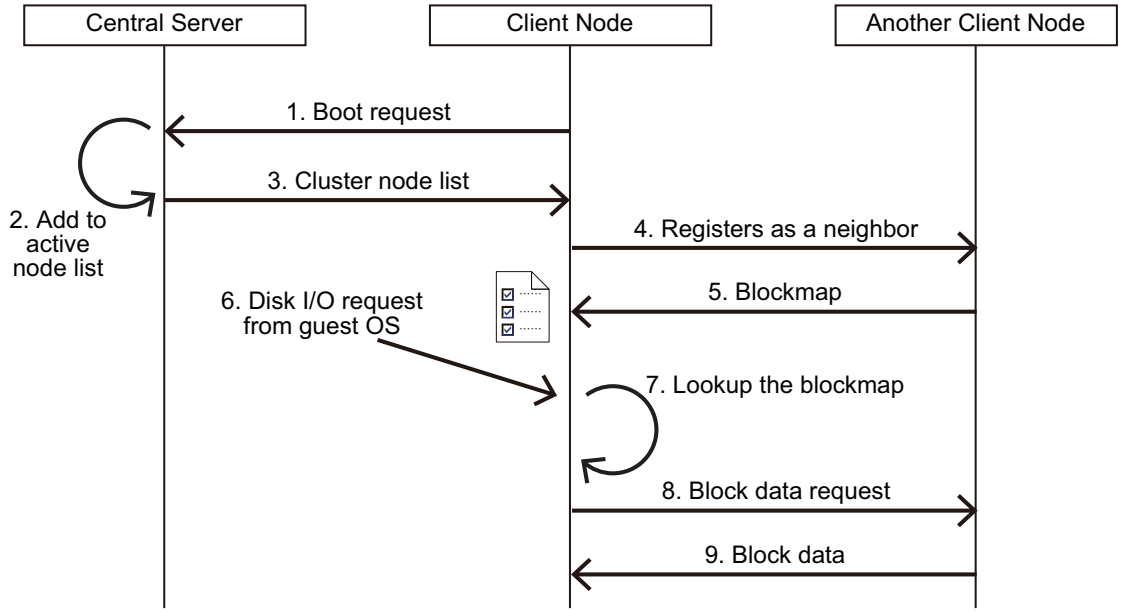


図 3.3: P2P 方式によるディスクイメージ配信システムにおける処理の流れ

を**利用者用環境**と呼ぶ。P2P ディスクイメージプロバイダは、この利用者用環境を格納したディスクイメージをネットワーク経由で利用可能とする。

システムの利用者がクライアントの電源を投入すると、まずホスト OS がローカルディスクから起動する。次に、ホスト OS は利用者に対し、起動可能な OS のリストを提供し、利用者はその中から望ましいものを選択する。前章で述べた起動・終了制御システムの場合、認証デバイスを用いて認証を行い、起動可能な OS を選択する。起動処理における、中央サーバとクライアントノードの処理の流れを図 3.3 に示す。利用者が OS を選択すると、P2P ディスクイメージプロバイダは中央サーバから**クラスタノードリスト**を取得する。クラスタノードリストとは、同一クラスタに所属し、ネットワーク的に近いノードの情報を格納したリストである。P2P ディスクイメージプロバイダはその中からいくつかのノードを選び、**ネイバーリスト**に加え、ブロックマップ要求を送信する。**ブロックマップ**とは、あるブロックをどのノードが持っているかを格納したメタ情報のリストである。ブロックマップの取得が終わると、P2P ディスクイメージプロバイダは仮想計算機を起動する。

仮想計算機として動作する利用者用環境がディスクブロックにアクセスすると、そのアクセスは仮想計算機モニタにより処理される。P2P ディスクイメージプロバイダはその処理をフックし、要求されたブロックのデータを提供する。もしそのブロックを持っていない場合には、ブロックマップの情報を利用して、他のノードや中央サーバにブロック要求を送り、ブロックのデータを受け取る。P2P ディスクイメージプロバイダは、このブロックマップの情報を定期的にノード間で交換する。

利用者が OS を停止させると、P2P ディスクイメージプロバイダはそれを検出し、中央サーバに対して OS が終了したことを伝える。中央サーバはクラスタノードリストを更新し、それをクライアントに通知する。

本研究の手法では、性能の観点から、ネットワーク的に近いノードを**ネイバーノード**として登録し、ネイバーノード間で情報やブロックの交換を行う。ネットワーク的に近いノードとは、たとえば、同じネットワークスイッチに接続されているノードを言う。このような場合、中央サーバからではなく、近いノードからディスクブロックを取得することがで

きれば、中央サーバの負荷とネットワークトラフィックを軽減することが可能である。

本研究の手法では、ある OS を起動しようとしているクライアントは、ある 1 つのクラスタに所属する。クライアントがどのクラスタに所属するかは、(1) どの OS を起動しようとしているか、(2) どの計算機を利用しているか、という情報をもとに、管理者が記述したルールにより自動的に決定される。ネットワークブートシステムが管理コストの削減に大きく寄与するような環境においては、そのネットワーク構成は合理的なサブネットに分割され、また計算機とネットワークスイッチは合理的に配置される。そのため、サブネットマスクやホスト名のルールなどを記述することによりクラスタを自動的に決定することができる。これにより、同一クラスタに所属するクライアントは全てネットワーク的に近いクライアントとなる。

3.3 P2P 方式配信システムの実装

本節では、前節で述べた機能を持つ P2P 方式のネットワークブートシステムの実装について述べる。

従来のネットワークブートシステムでは、まず、BIOS がブートローダを呼び出し、ブートローダが OS のカーネルを呼び出す。そして、OS のカーネルがネットワーク経由でルートファイルシステムにアクセスする。これには、何らかのネットワークプロトコルや、ネットワークブートシステムが持つ抽象化レイヤの機能を用いる。たとえば、NFS-root Linux や Mac OS X では、リモートサーバ上にあるルートファイルシステムに対し、NFS を用いてアクセスを行う。Citrix PVS 上で動作する Windows の場合では、PVS により提供されるディスク抽象化レイヤが Windows のローカルディスクアクセスをリモートサーバへのネットワークリクエストに変換することによりアクセスを行う。

本研究の手法では、まず、BIOS がブートローダを呼び出し、ローカルディスクからホスト OS として Linux を起動する。ホスト OS 上では、P2P ディスクイメージプロバイダが動作する。ホスト OS として Linux を用いることにより、P2P ディスクイメージプロバイダを通常の Linux 上のアプリケーションとして実装できること、既存のライブラリを利用できることといった利点が得られる。この P2P ディスクイメージプロバイダは KVM を用いて仮想計算機を作成し、その上でゲスト OS として利用者用環境を実行する。そのため、本研究の手法では、KVM 上で動作する OS であれば、Microsoft Windows を含めた様々な OS を稼動させることができる。

KVM は、ゲスト OS からのディスクアクセスを、その OS のディスクイメージへのファイルアクセスに変換する。これにより、本研究の手法では、従来手法でゲスト OS に必要となっていたネットワークプロトコルや抽象化レイヤを必要としない。このディスクイメージファイルへのアクセスは、P2P ディスクイメージプロバイダによりフックされる。これには、Linux においてファイルシステムを容易に実装することができる FUSE (Filesystem in userspace) と、この FUSE を Ruby から容易に利用することができる fusefs[1] を用いる。FUSE と fusefs を用いることにより、複雑なカーネルレベルのプログラミングを必要とせず、Ruby の持つライブラリを利用して実装を行うことができる。その一方で、カーネルレベルで動作する NFS に比べ、ユーザランドで動作することによるオーバヘッドが生じる。本研究では、クライアントノード上で動作する P2P ディスクイメージプロバイダと、中央サーバ上で動作するサーバプログラムの双方を Ruby を用いて実装した。

3.3.1 プロトコル

本研究での実装では、中央サーバ・クライアントノードにおける通信に、MessagePack-RPC[2][67] を用いている。これは、HTTP を利用したプロトコルを用いる場合に比べ、より高速・低レイテンシな通信を行えるという理由による。HTTP-FUSE KNOPPIX では、配信においてファイアウォールを容易に越えることができることや、CDN を利用して負荷分散を行うことができるという理由から、HTTP を利用している。本研究の手法においては、ファイアウォールを越えられることよりも、対象となるキャンパスネットワークが持つ高速・低レイテンシという特徴を活かすことが重要であるため、MessagePack-RPC を用いた独自プロトコルを利用する。

本研究の P2P 方式による手法を実現するため、次のような手続きを実装し、起動開始の宣言やメタ情報の交換、ブロックデータのやりとりを可能とする。ここで `server` は中央サーバを、`caller` は遠隔手続き呼び出しを行う側のノードを、`callee` は呼び出される側のノードを、`server_or_client` は中央サーバまたはクライアントノードを、それぞれ意味する。

`server.client_hello(os)`
: クライアントはサーバに対し、`os` で示される OS を起動しようとしていることを伝える。サーバは、クラスタノードリスト (その OS を起動している、同一のクラスタに所属するノードの一覧) を返す。

`server.client_bye(os)`
: クライアントはサーバに対し、`os` で示される OS を終了しようとしていることを伝える。

`callee.neighbor_hello(caller)`
: クライアントは、他のクライアントに対し、そのクライアントをネイバーリストに登録することを伝える。

`server_or_client.get_block(number)`
: クライアントは、サーバまたはクライアントに対し、`number` で示されるブロックのデータを要求する。サーバまたはクライアントは、当該ブロックのデータを返す。

`callee.blockmap_request()`
: クライアントは、他のクライアントに対し、ブロックマップ情報 (ブロックのメタ情報) を要求する。他のクライアントは、自らが持つブロックマップ情報を返す。

3.3.2 中央サーバとクライアント管理

中央サーバでは、`client_hello()` 手続きと `client_bye()` 手続きにより、現在 OS を実行しているノードのリストである**アクティブノードリスト**を維持する。アクティブノードリストは、次に示すように、クライアント ID と、そのクライアントが所属するクラスタの ID の組からなるリストである。

```
# Active Node List:
< client ID, cluster ID >
< client ID, cluster ID >
...
```

client_hello() 手続きが呼び出された場合、サーバは起動しようとしている OS の情報と、あらかじめ設定されたルールを用いて、所属するクラスタを決定する。次に、そのクライアントの ID とクラスタの ID をアクティブノードリストに加え、アクティブノードリストの中から、同一クラスタに所属する他のノードの情報をクライアントに返す。これを受け取ったクライアントは、その中からいくつかを選び出し、そのノードに対して neighbor_hello() 手続きを呼び出す。このようにして、起動を開始しようとしているノードは、既に同じ OS を起動しているノードの情報を得ることができる。client_bye() 手続きが呼び出された場合、サーバはそのクライアントのエントリをアクティブノードリストから取り除く。

一般的に、性能向上と高可用性のために、サーバを複数実行させられることが望まれる。本研究の手法においても、中央サーバを多重化することにより、性能と可用性を向上させることができる。その場合、クライアントノードは、起動開始時や中央サーバへのブロックデータ要求時に、利用可能な中央サーバ群のサーバの中からランダムに 1 台を選び、リクエストを行う。中央サーバにおいては、この多重化した状態においても適切に動作するため、アクティブノードリストの情報をサーバ間で同期させる。これには、サーバ間でのみ利用される cluster_join()、cluster_leave() 手続きが使用される。

3.3.3 クライアントにおけるブロックマップ管理とブロックデータの扱い

クライアントノードにおいては、ディスクイメージのメタ情報であるブロックマップを用いて、どのブロックに対してどのノードへリクエストを送るべきかを決定する。このブロックマップを更新するため、クライアントノードは自らのネイバーノードに対して blockmap_request() 手続きを呼び出し、そのノードが持つブロックマップ情報を取得する。この手続きが呼び出されると、クライアントノードは自らが持つブロックの情報を返す。このとき、自らが持つブロックの情報のみを返すことも、それに加えて自らが知っているブロックの所在を含めて返すこともできるが、本実装では前者の方法とした。これは、後者の方法を用いることにより、ブロックの所在をより素早く伝搬させることができると考えられるが、いくつかの実験の結果から、素早く伝搬させることができるメリットよりも、このブロックマップ情報の作成・統合の処理のオーバヘッドというデメリットの方が大きかったためである。本実装においては、クライアントノードは次のようなブロック番号のリストを返す。

```
# Blockmap information
< block_number, block_number, ... >
```

ゲスト OS からのディスクアクセスが生じた場合、P2P ディスクイメージプロバイダは、要求されたブロックがキャッシュ上に存在するかを確認する。存在する場合、そのデータをゲスト OS に返す。存在しない場合、ブロックマップを参照し、そのブロックを持っているノード群の中から任意のノードを選び、get_block() 手続きを呼び出し、ブロックデータを取得する。何らかの理由により正しくデータを取得できなかった場合、速やかに他のノードに対して手続きを呼び出し、処理を続行する。もしそのブロックを持っているノードがいない場合、中央サーバ群から任意のサーバを選び、手続きを呼び出し、処理を続行する。また、あらかじめ 1 台だけが起動している状態において、一斉に多数のクライアントが起動を開始したような場合を考えると、そのあらかじめ起動していたノードに対して多数のクライアントがリクエストを送る状態に陥ってしまう。これを避けるため、クライ

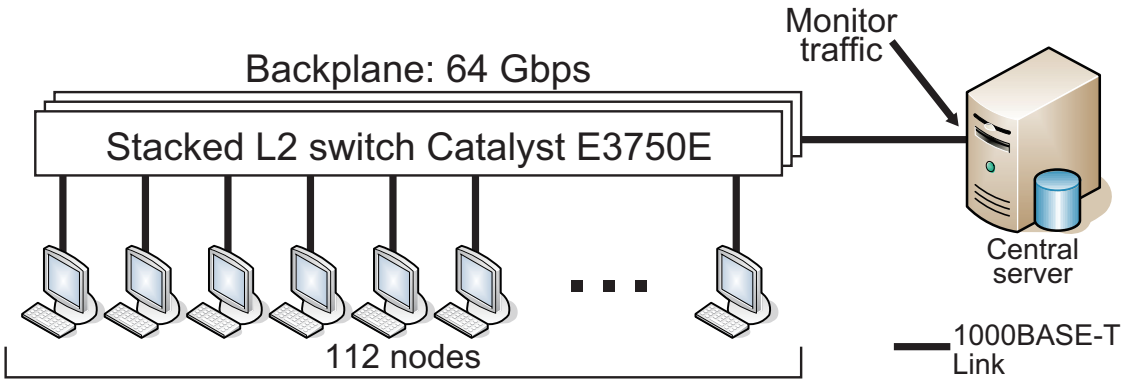


図 3.4: 112 台のクライアントを用いた実験環境の概要

アントノードにおいては、あるブロックを持っているノードが存在する場合でも、一定の確率でサーバに対してリクエストを行うようにしている。

3.4 評価

本節では、本研究で提案する P2P 方式のネットワークブートシステムの評価実験とその結果について述べる。評価実験は 112 台のクライアントを用いて行い、従来の NFS による方式と、本研究で提案する P2P 方式について、その起動所要時間とトラフィックについて測定、比較する。

3.4.1 実験環境

評価実験を行う環境を図 3.4 に示す。この実験は、筑波大学の実習室を用いて、112 台のクライアントノードを用いて行う。この実習室に中央サーバを 1 台接続し、P2P 方式の配信サーバないし NFS サーバとして設定し、実験を行う。各クライアントノードと配信サーバのスペックは、表 3.1 に示すとおりである。なお、後述する帯域幅の影響を測定する実験においては、表 3.1 中の配信サーバ 2 を、それ以外の実験においては、同表中の配信サーバ 1 を、それぞれ配信サーバとして用いる。

これらのクライアントノードと配信サーバは、スタック接続された Cisco Catalyst 3750E スイッチに接続される。このスイッチは 64Gbps のバックプレーン容量を持つ。

配信サーバでは、Linux の Proc ファイルシステムにより提供される `/proc/net/dev` ファイルを用いて、各 NIC におけるトラフィックを計測する。配信サーバの帯域幅を変更して測定を行う実験においては、測定対象とする全ての NIC のトラフィックをこのファイルを用いて計測する。なお、本実験は筑波大学において実際に運用されている実習室を用いて行うものである。そのため、帯域幅を変更する実験において、複数本のリンクを束ねて帯域幅を増やす、いわゆるトランキングの設定を行うことができない。代替策として、異なるサブネットに属する IP アドレスを割り当てたネットワークインタフェースを作成し、それぞれを 1Gbps のリンクでスイッチに接続することにより 1Gbps×n 本のリンクを用意し、これを用いることとした。

各クライアントノードでは、仮想計算機モニタ KVM を用いて仮想計算機を作成し、その上で利用者用環境として Microsoft Windows XP SP3 を起動させる。この OS は QCoW2

(QEMU Copy on Write) フォーマットのディスクイメージに格納され、その仮想サイズは 10GB、ディスクイメージファイルの実際のサイズは 4.6GB である。

3.4.2 測定と起動タイミング

このような実験環境を用いて、本研究で提案する P2P 方式の手法と、従来の NFS の手法についての実験を行う。実験においては、クライアントノードが起動を開始してから起動を完了するまでにかかる時間 (**起動所要時間**) と、その起動処理における配信サーバのトラフィックを測定する。

なお、「起動を開始」した時刻とは、クライアントノードにおいて起動処理を行うスクリプトを実行した時刻をいい、「起動を完了」した時刻とは、Windows の起動処理のうち、自動ログインが設定されたユーザの「スタートアップ」に登録された Internet Explorer が起動された時刻をいう。測定対象の Windows には、自動ログインが設定されているため、対話的なログイン処理なしに、あらかじめ設定されたユーザのデスクトップが表示される。このユーザにおいて、OS の「スタートアップ」項目に Web ブラウザである Internet Explorer を登録する。また、ブラウザの起動時に表示される URL として、ホスト OS 上で動作する CGI プログラムを指定する。これらの設定により、ゲスト OS が起動すると、自動的にホスト OS 上の CGI プログラムが実行されることになる。CGI プログラムがその時刻を記録することにより、起動完了時刻を計測することができる。これにより、多数のクライアントノードを対象とした実験においても、自動的に起動所要時間を測定することできる。

このようにして、(1) 起動処理を行うクライアントノードの台数、(2) P2P の場合のブロックサイズと、(3) 起動開始タイミング、の 3 種類のパラメータを変更しながら、起動所要時間とトラフィックの測定を行った。(1) のノードの台数は、15 台、30 台、61 台、112 台とした。これらはそれぞれ、小規模から大規模な授業の受講者を想定している。(2) のブロックサイズは、P2P 方式において、そのブロックサイズがどのように影響を与えるのかを調べるものである。(3) の起動開始タイミングは、(1) で設定したクライアントノード群が、どのようなタイミングで起動処理を開始するかを決定するものである。

本研究では、大学の実習室のような環境を対象としている。そのため、授業開始時や教員の指示などのタイミングに合わせて起動処理が実行されるようなケースが想定される。このような場合では、電源の投入は人間の手によって行われる。そのため、ある時刻を基準として、ある時間幅の中で、その起動開始処理が分布することになる。

ここで、筑波大学 情報科学類の実習室における、利用者の起動開始処理の分布を図 3.5 に示す。X 軸は時間を、Y 軸は累計起動台数を表す。図中の四角は、実習室における実際の起動開始処理の分布を表している。図中の曲線は、正規分布の累積分布関数を表している。この図から、実際の実習室における起動開始処理の分布が正規分布に従うことが分かる。

今回の測定においては、この正規分布に従った起動開始処理の分布について、その時間幅を 0 秒、15 秒、30 秒、60 秒とする。特に、時間幅 0 秒とは、ある基準時刻において全てのクライアントが一斉に起動処理を開始するケースである。また時間幅 15 秒とは、正規分布に従った遅延のもとで、基準時刻から 15 秒以内に全てのクライアントが起動処理を開始するケースである。

3.4.3 NFS における測定

従来の NFS による測定は、以下のような環境を用いて行う。まず、表 3.1 における配信サーバ 1 上で NFS サーバを稼動させる。本実験では、NFS のプロトコルとしてバージョン

表 3.1: クライアントノード・配信サーバのスペック

クライアントノード	
CPU	Intel Core2Duo E8500 (dual core)
Memory	4096 MB
HDD	Serial ATA 80 GB
OS	CentOS 5.8 x64 (Linux 2.6.18)
NIC	1000 BASE-T (1Gbps)
VMM	KVM 83.224.el5
配信サーバ 1	
CPU	Intel Xeon X3440 (quad core)
Memory	8192 MB
HDD	SAS 600GB
OS	CentOS 5.6 x64 (Linux 2.6.18)
NIC	1000 BASE-T (1Gbps)
配信サーバ 2	
CPU	Intel i7-3820 (quad core)
Memory	16384 MB
HDD	Serial ATA 500GB
OS	CentOS 5.6 x64 (Linux 2.6.18)
NIC	1000 BASE-T (1Gbps, quad port × 2)

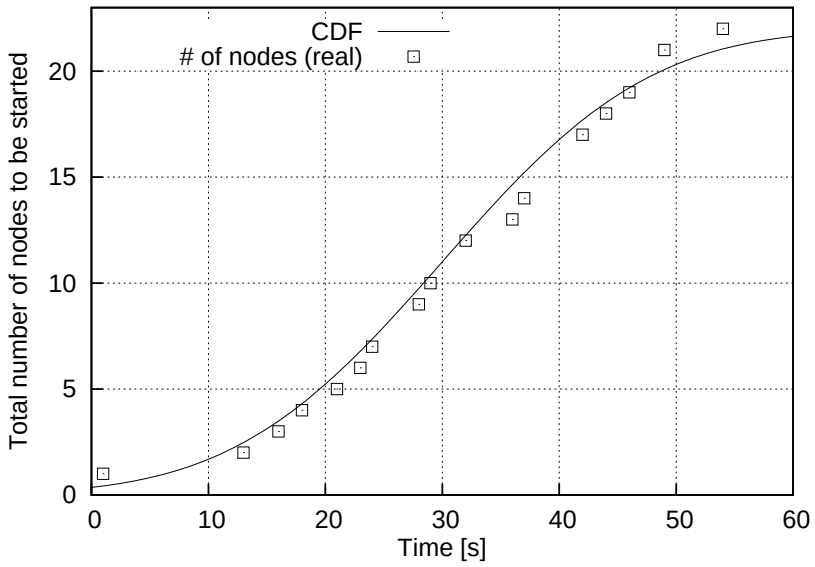


図 3.5: 起動タイミングと正規分布

4 を用いた。この NFS サーバ上に、実験で用いる利用者用環境を格納したディスクイメージを設置し、そのディレクトリを NFS で export した。

クライアントは、NFS サーバをマウントすることにより、ディスクイメージを読み込むことができる。OS の動作に伴う書き込み処理を適切に取り扱うために、起動処理を開始する前に、QCoW2 フォーマットの持つ差分ディスクイメージ機能を用いる。この機能により、読み込みは NFS サーバ上のデータから行うが、書き込みが行われる場合には、そのデータのみを別のファイルに書き込むことができる。測定開始前にこの差分ディスクイメージを作成し、仮想計算機のディスクイメージとしてこの差分ディスクイメージを指定する。これにより、他のクライアントからの書き込みの影響を受けることなく、OS の起動を行うことができる。

帯域幅を変更する実験においては、表 3.1 における配信サーバ 2 を NFS サーバとして同様に設定する。各 NIC にはそれぞれ異なるサブネットの IP アドレスが割り当てられており、全ての IP アドレスで NFS サーバにアクセス可能となるよう設定する。クライアントは、自らの IP アドレスをもとにして、サーバへのアクセスに用いるべきサブネットを決定し、その IP アドレスを用いてマウントを行う。

3.4.4 P2P 方式における測定

本研究で提案する P2P 方式による測定は、以下のような環境を用いて行う。まず、表 3.1 における配信サーバ 1 上で中央サーバを稼動させる。この中央サーバ上に、実験で用いる利用者用環境を格納したディスクイメージを設置し、配信システムの配信対象として登録した。

クライアントは、P2P ディスクイメージプロバイダを用いることにより、配信サーバ上のディスクイメージにアクセスすることができる。OS の動作に伴う書き込みは、P2P ディスクイメージプロバイダにより、別のブロックファイルへ書き込まれる。これにより、他のクライアントノードからの要求に対しても、適切なデータを返すことができる。

帯域幅を変更する実験においては、表 3.1 における配信サーバ 2 を中央サーバとして同様に設定する。各 NIC にはそれぞれ異なるサブネットの IP アドレスが割り当てられており、全ての IP アドレスで中央サーバにアクセス可能となるよう設定する。クライアントは、自らの IP アドレスをもとにして、サーバへのアクセスに用いるべきサブネットを決定し、その IP アドレスを用いて P2P ディスクイメージプロバイダの初期化処理を行う。

3.4.5 起動台数が与える影響

まず、起動タイミングを時間幅 60 秒に固定し、起動台数を 15 台から 112 台まで変えたときの起動所要時間をグラフに表したものを図 3.6 に示す。X 軸は起動台数を、Y 軸は各測定結果の中央値を秒単位で表している。

グラフから、まず NFS の測定結果が非常に線形なものであることが分かる。これは起動台数が増加するに従って起動所要時間の中央値も増加していることを意味する。一方、P2P 方式におけるブロックサイズを 512KB, 1024KB とした場合の測定結果では、同様に起動台数の増加に対して起動所要時間の中央値が増加するが、その増加の割合は NFS の結果に比べて緩やかである。また、ブロックサイズを 64KB, 128KB, 256KB とした場合の測定結果は、台数の増加の影響をほとんど受けていないことが分かる。これは、ブロックサイズが大きい場合、1 ブロックのうち本当に必要とされていた部分の割合 (占有率) が小さくな

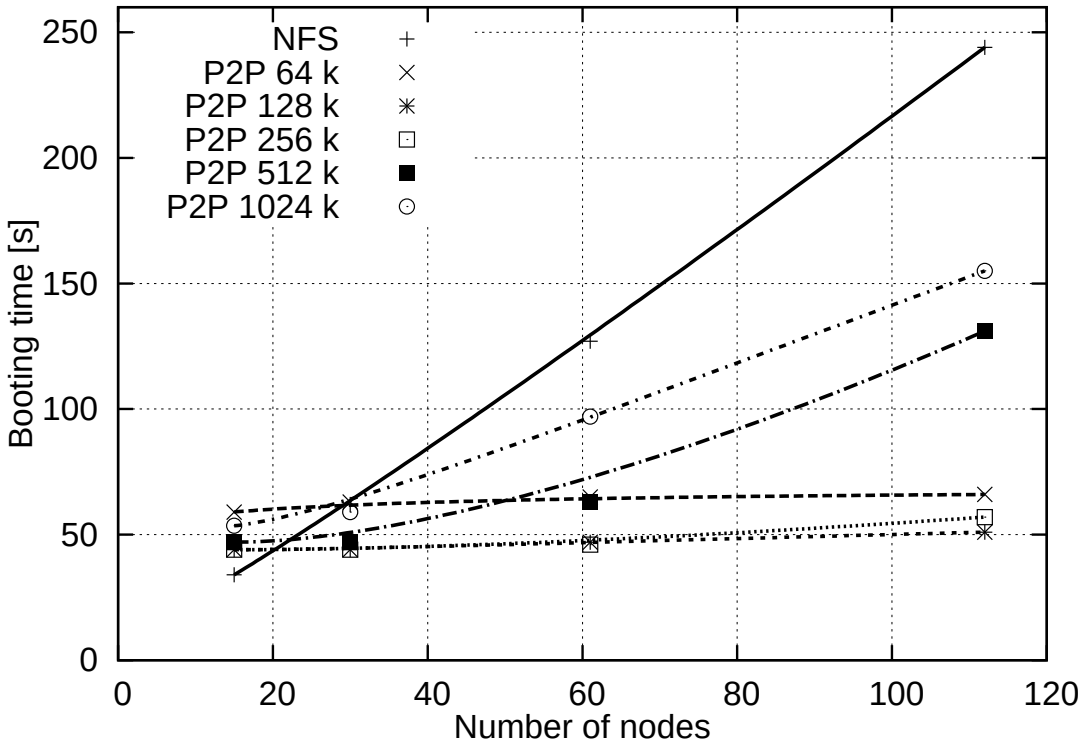


図 3.6: 起動台数を変化させた場合の測定結果

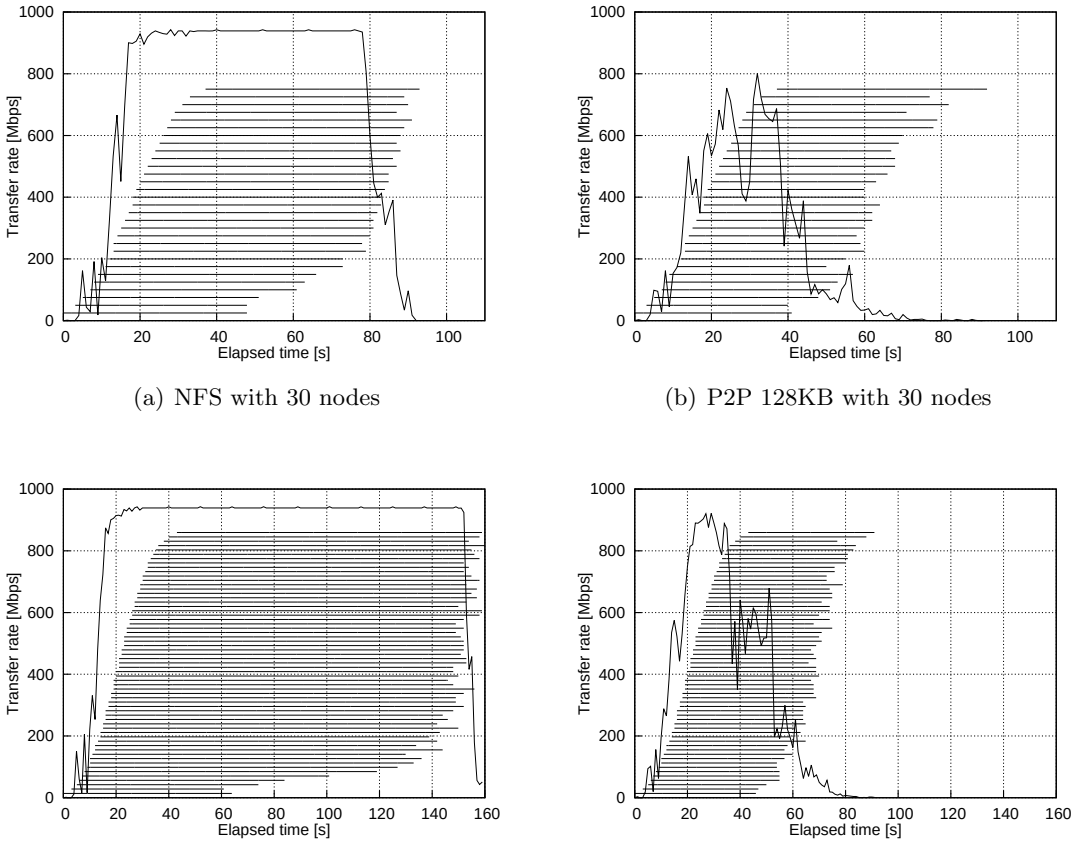
り、転送処理のオーバヘッドが大きくなることに起因すると考えられる。逆に、ブロックサイズを小さくした場合、占有率は大きくなるが、取り扱うべきブロックの量が増えるため、RPC の回数増加によるオーバヘッドが生じる。

起動処理を完了するために必要なデータのサイズを測定するために、fusefs を用いて `read()` 要求の記録を行うプログラムを作成した。このプログラムを使用して測定時と同様の仮想計算機を作成し、記録を行った。その結果から、起動処理を完了するために必要なデータの総量を計算すると、174MB であった。それに対し、P2P 方式の各ブロックサイズの場合における転送量と占有率を表 3.2 に示す。ブロックサイズ 128KB の場合では占有率は約 56%だが、256KB の場合では約 40%、512KB では約 24%まで低下する。このように、ブロックサイズは転送量に大きく影響を与える。

次に、この測定により得られた中央サーバのトラフィックのグラフを図 3.7 に示す。そ

表 3.2: 転送量と占有率

Block size [KB]	Traffic [MB]	Utilization ratio [%]
(FUSE)	174	(100)
128	308	56
256	446	40
512	717	24
1024	1258	14



(a) NFS with 30 nodes

(b) P2P 128KB with 30 nodes

(c) NFS with 61 nodes

(d) P2P 128KB with 61 nodes

図 3.7: 中央サーバのネットワークトラフィック

れぞれのグラフにおいて、X 軸は経過時間を秒単位で、Y 軸は中央サーバのトラフィックを Mbps 単位で表す。また、図中の水平線は、各クライアントノードの起動開始時刻と起動完了時刻を表す。図 3.7(a), 図 3.7(c) は、30 台の場合と 61 台の場合の、NFS による方式におけるトラフィックである。NFS サーバのトラフィックは、起動処理開始後に帯域幅の限界近くまで到達し、起動処理が終了するまで高いまま維持されている。つまり、NFS の場合においては、NFS サーバの帯域幅がほぼ全て使われている状態にあるといえる。

これに対し、図 3.7(b), 図 3.7(d) は、30 台の場合と 61 台の場合の、ブロックサイズを 128KB とした P2P 方式におけるトラフィックである。NFS の場合と同様に、中央サーバのトラフィックは起動処理開始後に高い値を示すが、時間の経過に従い、そのトラフィックは少なくなっていることが分かる。これは、時間の経過に伴い、中央サーバの代わりにブロックを提供することができるノードの数が増えていることを意味する。ブロックを提供することができるノードの数が増えることにより、中央サーバが転送しなければならないデータの量が減少し、そのトラフィックと負荷を軽減することができていることがグラフから読み取れる。

この 61 台を対象とした測定において、配信サーバのトラフィックに着目すると、NFS の場合、1 台あたり 271MB のデータを転送する必要があった。つまり、61 台を起動するためには、配信サーバは 16,531MB を転送する必要がある。これに対し、ブロックサイズを 128KB とした P2P 方式では、配信サーバのトラフィックは 3,973MB であった。この結果は、配信サーバだけでなくクライアントノードもブロックの配信を行うことにより、配

信サーバのトラフィックを軽減することができていることを示している。なお、このとき、クライアント間でのトラフィック総量は約 14GB であった。配信サーバのトラフィックとあわせると約 18GB となり、NFS の場合よりも大きくなる。これは、表 3.2 でも示したとおり、ブロックサイズによるオーバヘッドの影響である。ブロックサイズ 128KB の場合、1 台あたりの起動に必要なデータ量は 308MB であった。NFS の場合は 271MB であることから、ブロックサイズを 128KB とすることにより生じる NFS の場合に対するオーバヘッドは 1.10 倍である。61 台を起動する場合においても、NFS では約 16GB であるのに対し、128KB の場合では約 18GB となったが、これは NFS の約 1.10 倍であり、1 台の場合に等しい。

3.4.6 起動タイミングが与える影響

前項の測定結果から、P2P 方式においては、先行して起動しているノードの数に応じて配信サーバのトラフィックを軽減することができることがわかった。この先行して起動しているノードの数は、起動を開始するタイミングに影響される。前項の実験においては、起動を開始するタイミングを、60 秒の時間幅を持つ正規分布に従ったものとして測定した。本項では、この時間幅を変更することにより、P2P 方式における「先行して起動しているノードの数」を変化させ、起動所要時間やトラフィックにどのような影響を与えるかを測定する実験について述べる。

この実験では、起動対象となるクライアントの台数を 61 台に固定する。その上で、起動を開始するタイミングを、時間幅 15 秒、時間幅 0 秒と変化させた場合の起動所要時間とトラフィックを測定する。

この測定におけるパラメータである時間幅とは、ある基準時刻からどの程度の幅を持って起動開始時刻がばらつくかを意味する。授業開始にあわせて起動処理を開始する場合では、授業開始時刻が基準時刻である。時間幅 60 秒の場合、その基準時刻から前後 30 秒以内に全ての起動開始処理が実行されることを意味する。また、本項の実験においては、起動所要時間の中央値ではなく、**総所要時間**に着目する。総所要時間とは、最初のノードが起動を開始してから、最も遅いノードが起動を完了するまでの所要時間を言う。図 3.8 は、前項のトラフィック測定結果のグラフにおける総所要時間である。これは、授業の例においては、全学生の計算機が起動を完了し、教員が授業を進行することができるようになるまでの時間にあたる。この時間が長くなってしまうと、授業の進行を滞らせてしまうことになる。

クライアント台数を 61 台とし、時間幅を 15 秒、0 秒とした場合の測定結果を図 3.9 に示す。横軸は総所要時間を秒単位で表している。まず、それぞれの測定ケースについて着目し、時間幅 0 秒の場合と 15 秒の場合について比較すると、どのケースにおいても、時間幅を 15 秒とした場合の方が総所要時間が短くなっていることが分かる。ここで、時間幅 0 秒の場合では、全てのノードが同時に起動を開始するため、最も起動に時間がかかったノードの起動所要時間が総所要時間となる。しかし、時間幅 15 秒の場合では、その時間幅のなかで一定時間待ってから起動を開始するノードが存在する。つまり、総所要時間にはこの待ち時間が含まれている。にもかかわらず、この測定結果は、時間幅 0 秒の場合に比べ時間幅 15 秒の場合の方が総所要時間が短くなることを示している。これはつまり、起動処理が集中することにより、総所要時間は悪化することを意味している。

時間幅を 15 秒としたことにより、P2P の場合でも NFS の場合でも、その結果は改善された。ここで NFS の場合、時間幅 0 秒の時に 154 秒だったものが、時間幅 15 秒では 153

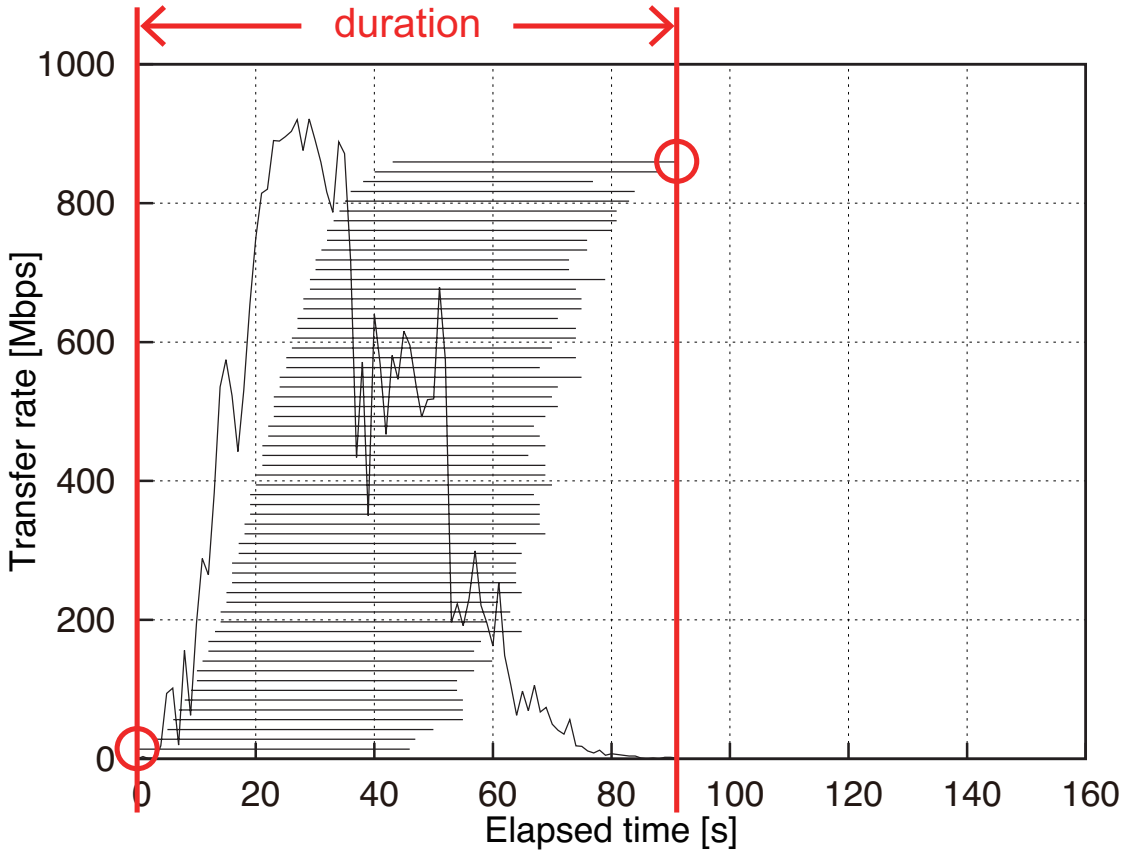


図 3.8: P2P 128KB, 起動対象 61 台, 時間幅 60 秒の場合における 総所要時間 (duration) の範囲

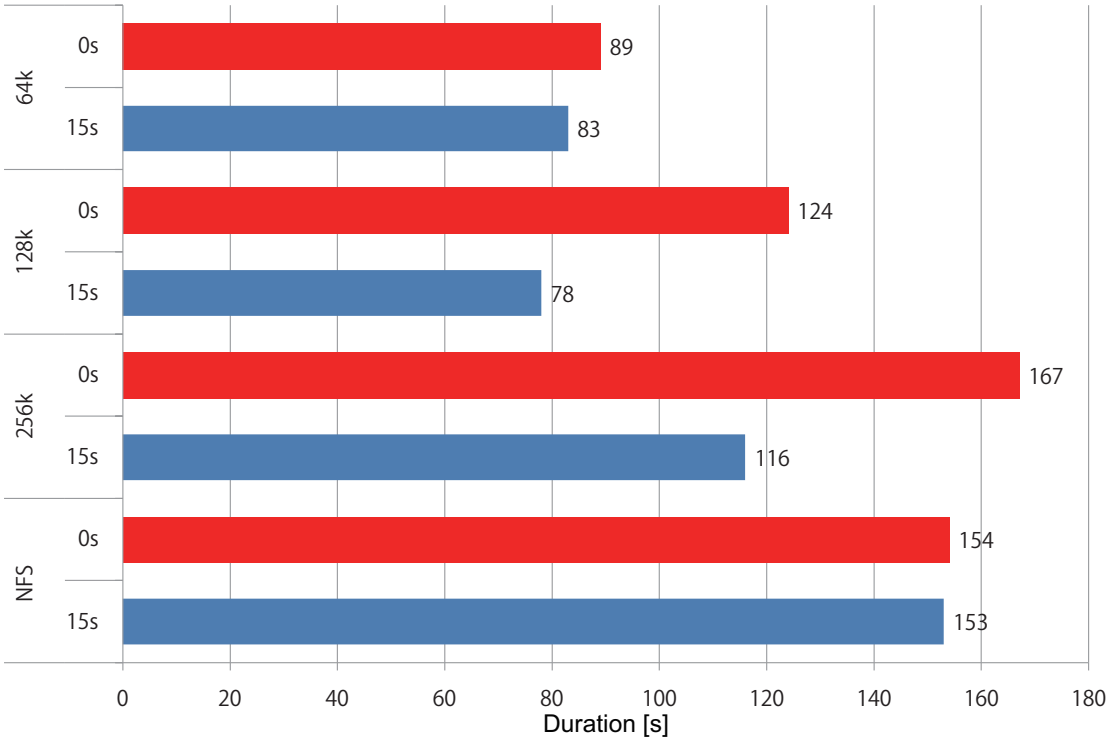


図 3.9: 時間幅 15 秒, 0 秒の場合における総所要時間計測結果

秒となった。これに対し P2P の場合では、64KB の場合には 89 秒から 83 秒に、128KB の場合は 124 秒から 78 秒に、256KB の場合は 167 秒から 116 秒に、それぞれ NFS の場合よりも大きく改善していることが読み取れる。これは、時間幅を広くし、起動処理の集中を緩和することが与える影響が、NFS の場合よりも P2P の場合の方が大きいことを意味する。NFS の場合では、起動処理の集中が緩和されたとしても、NFS サーバが配信すべきトラフィックの量は変化しない。P2P の場合、起動処理の集中が緩和されることで、後から起動するノードが他のノードからブロックを取得できる確率が向上する。それにより中央サーバが配信すべきトラフィックの量が削減され、総所要時間を改善することができる。

この実験結果から、P2P 方式においては特に、起動処理の集中を緩和することが重要であることがわかる。そこで、一斉起動時において、配信サーバが自動的に遅延をいれることにより、その総所要時間を改善することを考える。時間幅 15 秒の測定では、あらかじめ時刻 t_i にノード i が起動を開始するように設定を行っていた。しかし、実際にネットワークブートシステムとして稼働させる場合、時刻 t_i において時刻 t_{i+1} にどれだけクライアントノードが起動を開始するかを知ることができない。そのため、中央サーバがクライアントに対し起動処理の殺到を伝え、起動処理を遅延させるためには、それまでの起動処理の傾向から、これからの起動処理の発生を推定する必要がある。

起動処理の発生を推定し、総所要時間を短くできるような遅延を挿入するためには、次の点が重要となる。

1. 起動処理の殺到を緩和できるような遅延を挿入すること
2. 起動処理の殺到が起こっていない場合に、不要な遅延が挿入されないこと
3. 遅延の挿入が更なる起動処理の殺到を極力起こさないこと

これを満たすような遅延を挿入する方法について考える。

まず、起動処理の殺到とは、「単位時間あたりに起動処理を開始したクライアントの台数」であると考える。ここで、単位時間とは、1 台のクライアントが起動を完了するまでにかかる時間に比べて十分に短い時間とする。この台数が多ければ多いほど、あるクライアントが起動を完了する前に、次のクライアントが起動処理を開始する、つまり起動処理が殺到しているということになる。この台数の値を用いて、その起動処理の殺到状態を判断し、遅延を挿入することができる。この値が 0 台の場合、そのサーバはその瞬間において殺到状態にないと判断することができるため、そのような場合には遅延を 0 とすればよい。

次に、起動処理が殺到状態にある場合に、ほぼ同時刻に起動処理を開始しているクライアントに対してほぼ同じ遅延を挿入してしまうと、それらのクライアントは同じタイミングで起動処理を開始してしまう。このようなことが起きてしまうと、再びサーバに対してリクエストが殺到する状態に陥ってしまう。このような状態を極力回避できるように遅延を決定する必要がある。CSMA/CD (Carrier Sense Multiple Access/Collision Detection) [29] では、通信の衝突を検知した場合、再び衝突を起こす確率を下げるために乱数で決定した時間だけ待つ仕組みを持つ。この場合でも同様に、他のクライアントと同じ遅延になる確率を下げるために、遅延の決定において乱数を用いる。

本研究では、中央サーバは式 (3.1) に示すような遅延をクライアントに通知するようにし、クライアントを一斉起動した場合の、総所要時間を測定した。

$$delay_i = \alpha \cdot recent(t, span) + rand(\beta \cdot recent(t, span)) \tag{3.1}$$

ここで、$delay_i$ はノード i の遅延秒数を、$recent(t, span)$ は時刻 t における、$span$ 秒以内に起動を開始したノードの数を、$rand(n)$ は 0 から n までの値域を持つ乱数を意味する。$recent(t, span)$ は「単位時間 ($span$ 秒) あたりに起動処理を開始したクライアントの台数」であり、起動処理の殺到度合いを表す。$rand(n)$ は、遅延の挿入が再び殺到状態を引き起こすことを防ぐための乱数である。α, β はそれぞれ遅延の分布を決定するパラメータで、$[0, 1]$ の値域を取る。本実験では $\alpha = 1/8, \beta = 1/12$ とした。

中央サーバは、クライアントが起動を開始する際に、アクティブノードリストに起動開始時刻を保持する。新たなクライアントが起動を開始する際には、その時刻を用いて $recent(t, span)$ を計算し、クライアントが起動を開始する前に待つべき遅延を決定する。中央サーバがクライアントに対してクラスタノードリストを返す際に、サーバはこの遅延を通達し、クライアントはその時間だけ待ってから、起動処理を開始する。

この自動遅延挿入の仕組みを導入した環境において、61 台のクライアントが一斉に起動要求を行った場合の総所要時間を測定した。実験に際し、まず P2P 64KB のケースについて測定を行った。次に、そのときと同じ起動タイミングを用い、NFS のケースについて測定を行った。これを P2P 128KB のケースでも同様に行い、128KB の場合と同じ起動タイミングを用いて NFS のケースの測定を行った。この実験結果を図 3.10 に示す。なお、実験環境の都合から、この測定は表 3.1 における配信サーバ 2 を用いて、帯域幅 1Gbps の場合として測定している。

この結果から、P2P 方式の場合、逐次的な処理においても自動的に遅延量を決定することにより総所要時間を改善できていることが分かる。同じ遅延を挿入した場合において、NFS よりも P2P の場合の方が総所要時間を短縮することができているほか、時間幅 15 秒、60 秒の場合と比べてもほぼ同程度の総所要時間で起動を完了していることが分かる。

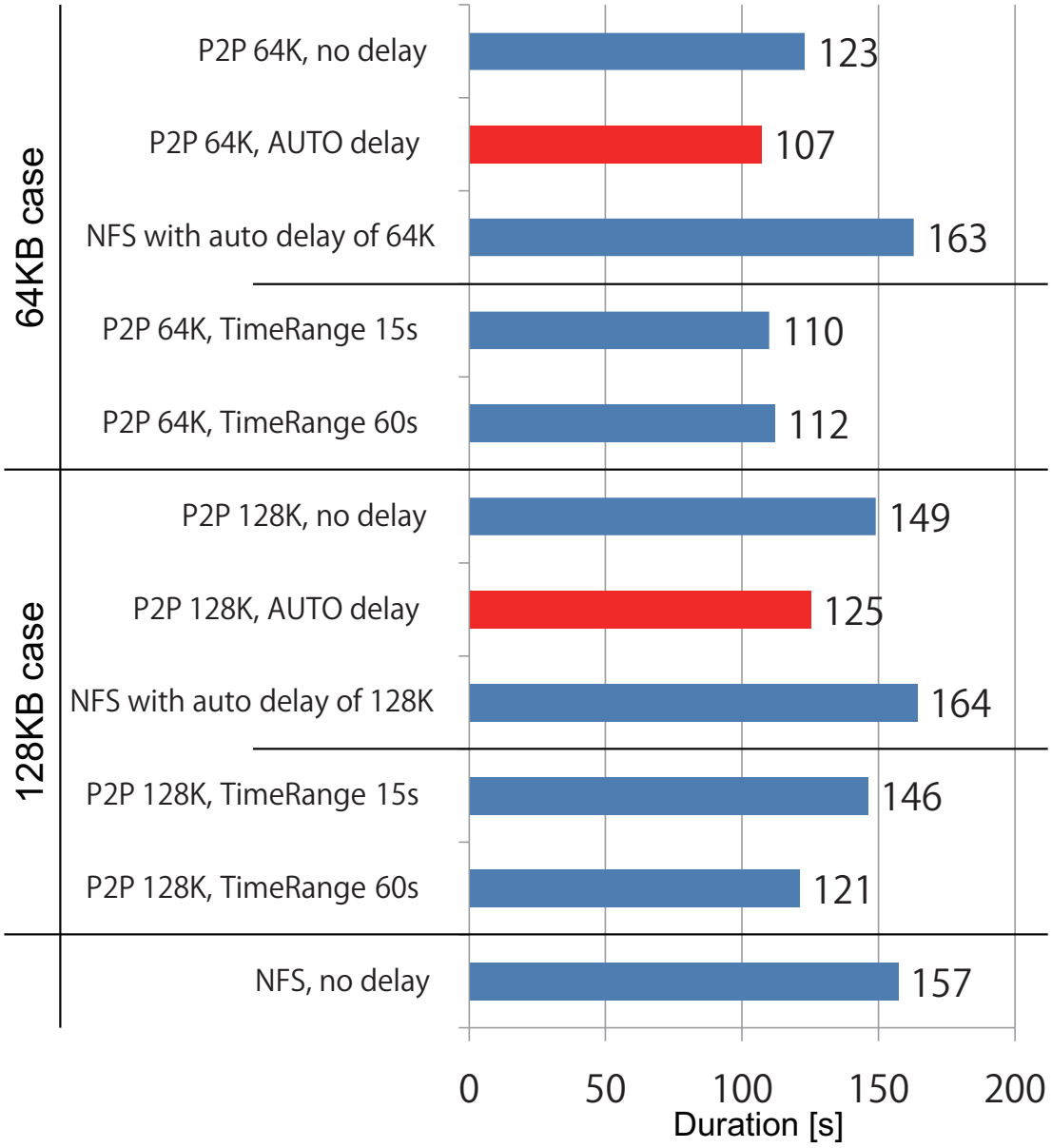


図 3.10: 遅延を自動的に挿入した場合の総所要時間測定結果

3.4.7 帯域幅の影響

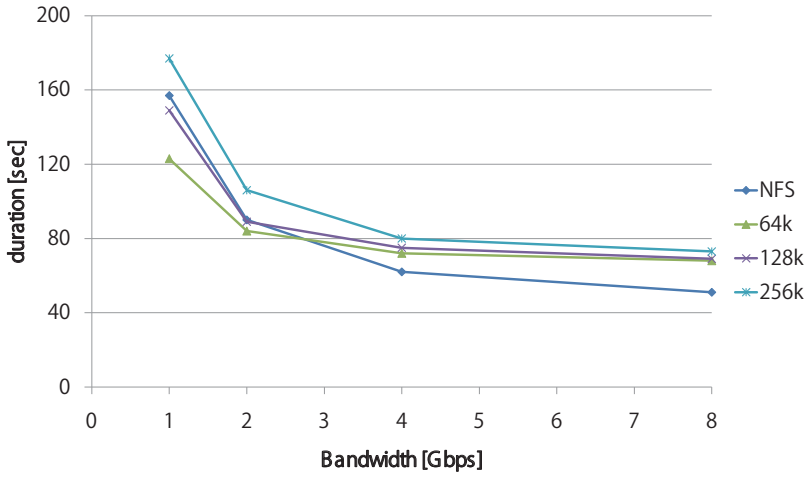
次に、中央サーバとして表 3.1 における配信サーバ 2 を利用し、中央サーバからスイッチまでの帯域幅を変更した場合にどのように総所要時間が変化するかを測定した。実験環境として大学の実習室を使う都合から、帯域幅を増やす方法としてトランキングを行うことはできないため、サーバからスイッチを 8 本の 1000BASE-T リンクで結び、それぞれに異なるサブネットの IP アドレスを割り当てた。クライアントは、自らの IP アドレスをもとにして、サーバへのアクセスの際に利用すべきサブネットを決定する。スイッチからクライアントまでのリンクは、これまで同様に 1000BASE-T リンクによる 1Gbps の回線とする。

このような環境で、帯域幅を 1, 2, 4, 8Gbps と変化させながら、時間幅を 0 秒, 15 秒, 60 秒と変化させた場合の総所要時間を図 3.11 に、時間幅を 15 秒とした NFS の場合の配信サーバのトラフィックを図 3.12 に、時間幅を 15 秒とした P2P 128KB の場合の配信サーバのトラフィックを図 3.13 に、それぞれ示す。

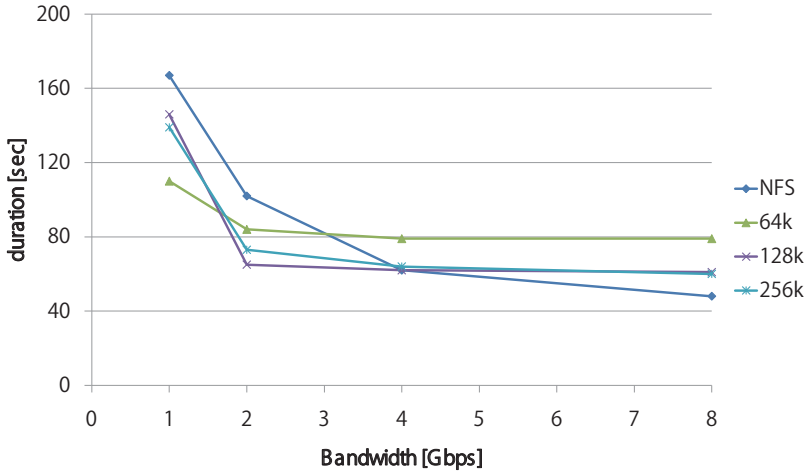
この測定結果から、NFS の場合においては、帯域幅を広くすることによりその総所要時間がほぼ線形に改善することがわかる。これは NFS の場合、8Gbps までの場合では、ネットワークの帯域幅がボトルネックになっていることを意味している。それに対して P2P の場合では、帯域幅を広くしたことの影響は時間幅によって異なっている。時間幅 0 秒 (図 3.11(a)) の場合では、NFS の場合と同様に、帯域幅を広くすることにより、その総所要時間が改善されていくことが分かる。一方、時間幅 60 秒 (図 3.11(c))、時間幅 15 秒 (図 3.11(b)) の場合、帯域幅を 1Gbps から 2Gbps に広げることにより、その総所要時間は大きく改善する。しかし、4Gbps 以上では、2Gbps の場合とほぼ変わらないという結果となった。

これは、P2P の場合、中央サーバからだけでなく、他のノードからもブロックを取得するという特徴に起因する。今回の実験で中央サーバの帯域幅を 8Gbps に広げた場合、中央サーバは 8 台のクライアントに対して、1Gbps の帯域幅を提供できることになる。そのため、リクエストが平等に分散される場合、中央サーバからスイッチまでは 8Gbps、スイッチからクライアントまでは 1Gbps の帯域幅を持つと仮定することができる。NFS の場合、全てのトラフィックが中央サーバから配信されるため、全ての通信が中央サーバからスイッチまでの帯域幅が広がったことによる恩恵を得られることになる。一方 P2P の場合、全てのトラフィックが中央サーバから配信されるとは限らず、他のノードが持っているブロックであれば、そのノードから取得することになる。そのため、中央サーバから配信されるトラフィックのみが、中央サーバからスイッチまでの帯域幅が広がったことによる恩恵を得られることになる。時間幅を大きくすることにより、後から起動するノードが、他のノードからブロックを取得できる確率が向上する。そのため、後から起動するノードは中央サーバからの配信を受けないため、帯域幅を広げた恩恵を受けにくくなる。

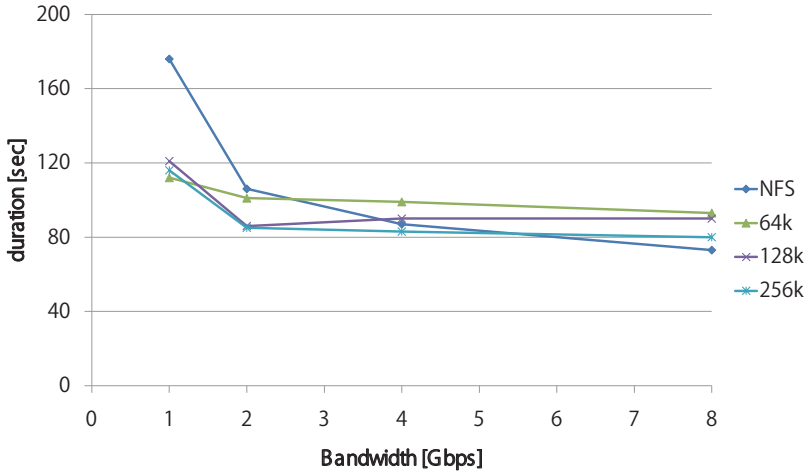
本研究の P2P による方式は、配信サーバが配信すべきトラフィックを、クライアントノード間でやりとりすることにより削減する仕組みである。図 3.12 に示す NFS の場合のトラフィックを見ると、4Gbps と 8Gbps の場合のそれぞれの後半部分を除き、その帯域幅をほとんど使い切っていることが分かる。つまり、サーバ側の帯域幅がボトルネックとなっている。一方図 3.13 に示す P2P 128KB の場合のトラフィックでは、2Gbps 以上の場合において、サーバ側の帯域幅を使い切っていないことが分かる。これは、クライアントノード間でのブロックのやりとりにより、サーバ側に必要な帯域幅を 2Gbps 程度まで抑えられていることを意味する。そのため、この測定の状態においてさらに起動台数を増やした場合、帯域幅を使い切ってしまっている NFS の場合に比べ、帯域幅に余裕のある P2P 方式の方がより多くのクライアントを処理できると考えられる。



(a) TimeRange: 0sec

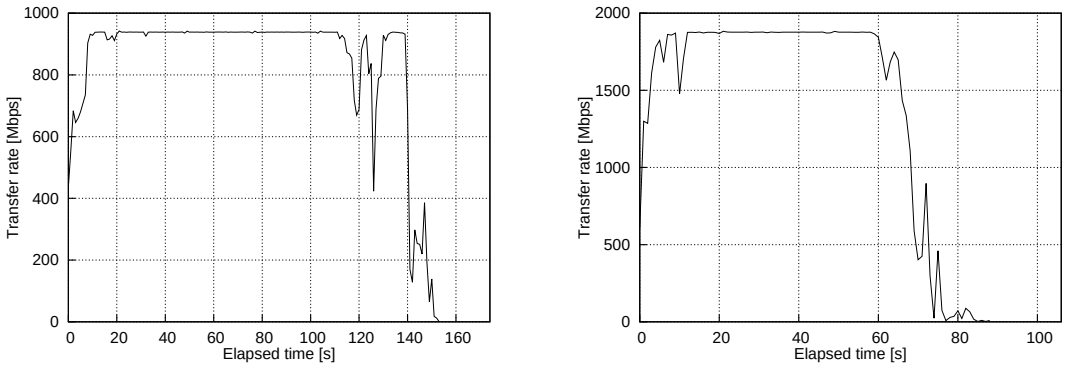


(b) TimeRange: 15sec



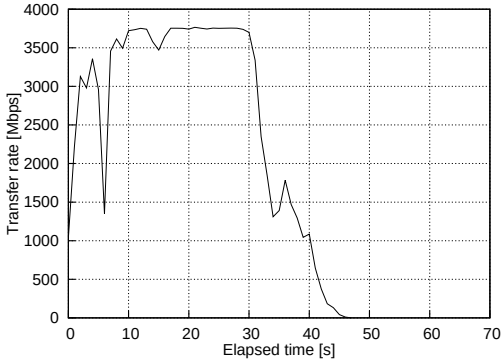
(c) TimeRange: 60sec

図 3.11: 中央サーバの帯域幅を変化させた場合の総所要時間

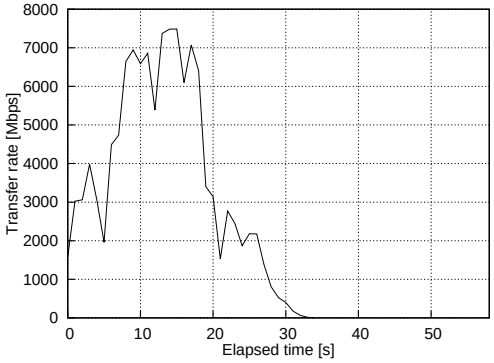


(a) NFS 1Gbps

(b) NFS 2Gbps

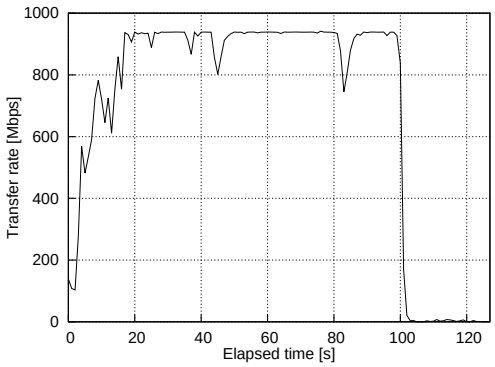


(c) NFS 4Gbps

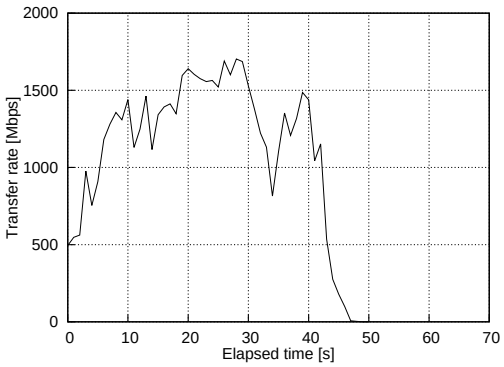


(d) NFS 8Gbps

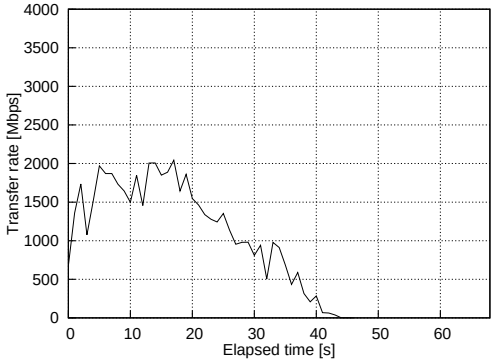
図 3.12: 時間幅 15 秒, NFS の場合の配信サーバのネットワークトラフィック



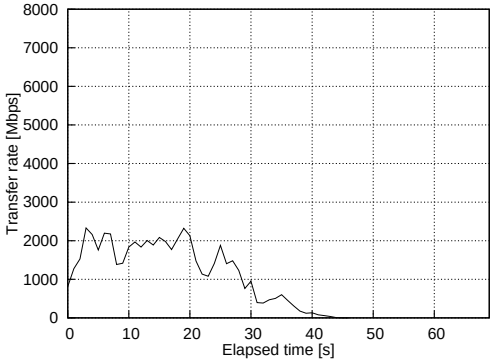
(a) P2P 128KB 1Gbps



(b) P2P 128KB 2Gbps



(c) P2P 128KB 4Gbps



(d) P2P 128KB 8Gbps

図 3.13: 時間幅 15 秒, P2P 128KB の場合の配信サーバのネットワークトラフィック

P2P 方式において、余剰分の帯域幅を活用してより高速に起動を完了するためには、配信サーバの帯域幅がどのような状態にあるかをクライアントに通知し、もし帯域幅に余裕がある場合には、積極的にサーバから取得を行うことが考えられる。3.3.3 項で述べたように、既にブロックの所在を知っている場合にも、負荷分散の観点から、一定の確率で配信サーバに対してリクエストを行う仕組みがある。この仕組みにおいて配信サーバの帯域幅の状態を加味することにより、サーバ側の帯域幅を有効活用できると考えられる。

このことから、本研究の手法は、配信サーバから末端のスイッチまでの経路において、その帯域幅を広くすることが容易でない場合に特に効果的であるといえる。たとえば、キャンパス全体に実習室が設置されているような場合や、遠隔地にキャンパスがあるような場合、配信サーバの置かれたサーバルームから広帯域の回線を引くことは容易ではない場合がある。このような場合においては帯域幅を広くすることが困難となり、NFS の場合、そのボトルネックを解消することが困難となる。それに対し本研究の P2P による方法では、経路上の回線が 1Gbps や 2Gbps 程度であっても、ボトルネックを解消し、高速に起動を完了することができる。

NFS の場合でも、遠隔地に NFS のミラーサーバを設置することにより、そのボトルネックを解消することが可能である。このミラーサーバの設置は、管理者にとって管理すべきサーバの数が増えることを意味し、またデータのミラーリングをどのように行うかという問題もある。本研究の P2P 方式の場合、起動を完了したクライアントノードが、NFS におけるミラーサーバの役割を担う。NFS の場合と異なり、ミラーサーバを明示的に設置することなく、クライアントノードが同等の振る舞いを行うため、多数のミラーサーバを設置することなく、ネットワークのボトルネックを解消することができる。

3.5 まとめ

本章では、ネットワークブートシステムに共通して存在する、OS イメージを配信する際のボトルネックの問題を、P2P 方式のディスクイメージ配信システムにより解決することについて述べた。先行研究として、シンクライアント OS として KNOPPIX を P2P 方式で配信する研究がある。この手法では、配信対象を Linux に限定し、その起動課程に手を加えることにより、クライアントの起動段階に応じたツリーを形成し、効率良くディスクブロックの配信を行う。それに対し本研究では、大学の実習室のような環境を対象とするため、多数の OS を多数の計算機に配信できることが必要となる。そのような環境では、授業開始時や教員の指示などに基づき多数の計算機が起動処理を開始するという状況が頻繁に起こりうる。本研究ではそのような場合に、その起動所要時間がどのように変化するかについて明らかにした。

OS イメージの配信において、クライアントが配信サーバからだけでなく、ブロックを既に持っている他のノードからも取得することにより、配信サーバの負荷やトラフィックを軽減する仕組みを設計、実装し、評価を行った。その結果、従来の NFS の方式に比べ、配信サーバのトラフィックが軽減され、クライアントの起動所要時間を改善できたことを示した。また、授業開始時や教員の指示などに基づき、多数の計算機がほぼ一斉に起動処理を開始するという状況を想定し、正規分布に従ってクライアントノードが起動を開始する場合における起動所要時間を測定した。その結果、従来の NFS の方式に比べ、本研究の手法の方が高速起動を完了できることを示した。また起動を開始するタイミングについて、P2P 方式の場合では、起動処理の集中を緩和することができれば NFS の方式よりもさらに高速に全てのクライアントが起動を完了できることを示した。加えて、配信サーバから末

端のスイッチまでの帯域幅を変化させた場合の影響についても測定を行い、帯域幅を広くすることにより、一斉起動のようなサーバの帯域幅がボトルネックとなる場合に、起動処理が高速化されることを示した。また、P2P 方式では配信サーバに求められる帯域幅を少なく抑えることができることを示し、遠隔地にキャンパスがある場合のように、配信サーバから末端までのネットワークを広帯域化することが難しい場合に、特に効果的であることを示した。

今後の課題としては、P2P 方式により生まれた配信サーバの余剰帯域幅を、配信サーバの状態に応じて活用することで起動処理を高速化する方法の検討や、FUSE と fusefs を用いることにより生じるオーバヘッドの軽減がある。

第4章　多数の認証システムを利用可能な URL レベル外向きアクセス制御

本章では、集中管理された利用者用計算機のうち、図書館における蔵書検索用の端末において、多数の認証システムを利用可能な URL レベル外向きアクセス制御について述べる。この仕組みを用いることにより、利用者は多数の認証システムの中から選択した認証システムを用いて認証を受け、Web アクセスを行うことを可能とする。

管理者が集中管理を行う計算機には、計算機実習室に配置された計算機の他に、図書館のような公共の場に設置される端末 (**キオスク端末**) がある。この端末は蔵書検索 (OPAC) 用の端末として提供され、Web 上の OPAC サービスを利用することができる。このような端末においては、管理者は利用者に自由な Web アクセスを認めず、利用者に応じて何らかのアクセス制御を行うことが多い。これは、組織の構成員だけでなく、部外者もその端末を利用するためである。そのため、インシデント対策として、アクセス先 URL などに基づいて OPAC サービスや図書館の情報ページ以外にアクセスできないような設定を行う。

キオスク端末の利用において認証を行い、組織の構成員であることが確認できた場合には自由な Web アクセスを許可することにより、そのような利用者にとっての利便性を向上させることができる。これにより、図書館で調べ物を行う際に、OPAC サービスの検索結果から著者の Web ページを閲覧するといったような利用が可能となる。このような利用を組織の構成員以外にも提供したい場合の例として、共同研究者が考えられる。共同研究者が大学を訪れた際に、図書館に設置された端末を用いて Web ページの閲覧や Web メールの送受信などが行えれば便利である。従来の方法では、このような利用者に対してもキオスク端末における自由な Web ページの閲覧を認めるためには、その利用者に対して一時アカウントを発行する必要がある。

この一時アカウントには、いくつかの問題点がある。まず、管理者から見た場合、一時アカウントの発行処理を行い、利用者に渡し、利用終了後に破棄する必要がある。また、利用者から見た場合、一時アカウントの発行申請を行い、ログイン情報を受け取り、それを管理する必要がある。つまり、利用者は管理すべきアカウントが 1 つ増えてしまう。このように、一時アカウントの発行には煩雑な手続きが必要となる。インターネットの普及により Web 上のサービスが発達し、日常生活において Web の利用が必要不可欠となっている現在では、Web を利用するためにこのような煩雑な手続きが必要となるのは望ましくない。

もし一時的アカウントを発行してもらうかわりに、共同研究者が既に持っているアカウントを用いて認証を受けることができれば、この問題を解決することができる。たとえば、管理者が Facebook による認証を認めている場合、この共同研究者が既に Facebook のアカウントを持っていれば、そのアカウントを用いて認証を受けることができる。これにより、管理者は一時的アカウントを発行する必要がなくなり、共同研究者は新たなアカウントを管理しなくて済む。

このとき、利用者が認証に用いることができる認証システムの数が多ければ多いほど、利用者にとっては使用可能なユーザアカウントを持っている可能性が高くなる。そのため、認証には多数の認証システムが利用可能で、かつ新たな認証システムに対しても容易に対応

できることが望ましい。また、認証システムから提供される利用者の情報 (**属性情報**) に基づいてアクセス制御のルールを記述できれば、より柔軟なアクセス制御を実現可能となる。

Web アクセス時の認証において、多数の認証システムを利用可能とする方法として、SPNEGO[65] がある。SPNEGO では、GSS-API[42] に対応した認証システムであれば、Kerberos[52] や NTLM[25] などを認証に用いることができる。しかし、属性情報やアクセストークンの受け渡しなどの処理が認証システムに依存してしまうなどの問題がある。また、Web アクセス時の認証において、シングルサインオンプロトコルの Shibboleth[32] を利用可能とする方法 [36] もある。この方法では、phantom URL と呼ばれる特別な URL を Web プロクシサーバが処理することにより、クッキーを用いて利用者の識別を行うことを可能としている。しかし、この方法では Shibboleth にしか対応できないこと、アクセス先ドメインの数だけ特別な URL によりクッキー発行処理が必要となることといった問題がある。

これらの、多数の認証システムに対応する際の属性情報の取り扱いの問題を解決するため、本研究では、**認証方法に依存しないユーザ識別子 (MI-UID)** と**ユーザ属性ストア**を用いて、属性情報を認証方法に依存しない形で統一的に取り扱う仕組みを提案する。認証システムに固有の認証処理は、対応する認証システムごとに用意される**ログインサーバ**が行う。これにより、新たな認証システムに対応する場合においても、ログインサーバを実装、設置するだけでよい。ユーザを識別する方法は MI-UID により統一されているため、そのユーザの属性情報を MI-UID に紐付けてユーザ属性ストアに格納することさえできれば、提案するシステムの認証システムとして用いることができる。

このようなアクセス制御システムを設計、実装し、その有用性を評価した。利用可能な認証システムとして Shibboleth[32] 認証に対応したアクセス制御システムを、筑波大学附属図書館において 182 台のキオスク端末を対象として運用し、その有用性を示した。また、新たな認証システムに対応する場合として、Facebook 認証に対応するために必要なコードの主要部分はわずか 33 行であり、容易に対応することができることを示した。

4.1 関連研究

本節では、Web アクセス制御と認証に関する研究について述べる。

4.1.1 ペアレンタルコントロール

Web の利用におけるアクセス制御を実現する方法のうち、通信を行うソフトウェア自身またはそのソフトウェアが動作する OS においてアクセスの可否を制御する方法として、ペアレンタルコントロールと呼ばれる機能がある。この機能を用いることにより、特定の利用者に対して、条件に基づいてアクセスを許可したり拒否したりすることができる。例としては、Microsoft Windows[15] や Mac OS X[7] では OS の機能として、Web ブラウザである Firefox[22] ではプラグインとして、提供されている。これらの方法では、その制御システムへのログイン方法として、OS のユーザアカウントやユーザ名とパスワードを用いたものなどが利用されている。本研究の手法では、多数の認証システムが利用可能であり、またその認証システムにより提供される属性情報に基づいたアクセス制御を可能としている。

4.1.2 様々な情報を利用した Web サーバにおける認証

Apache Web サーバでは、HTTP Basic 認証や Digest 認証 [23] により得られるユーザ名やグループ情報を用いて、アクセスコントロールルールを記述することができる [21]。このグループ情報に加え、Web サーバにおける認証方法として多数の認証システムを利用する仕組みが提案・実装されている。文献 [48] の方法では、RBAC(Role-Based Access Control) による利用者のロールに基づいた記述を可能とする。文献 [11], [19], [37] の方法では、X.509 証明書の属性を用いた記述を可能とする。文献 [10] の方法では、PCA(Proof-Carrying Authentication) による証明を用いた記述を可能とする。これらの方法はどれも、Web サーバにおいて利用者の認証とアクセス制御を行うものである。本研究の手法では、Web プロクシサーバにおいて、外向きのアクセス制御を行う。

4.1.3 外向きアクセス制御

図書館やホテルのような環境においては、外向きアクセス制御のために、Captive Portal と呼ばれる方法がしばしば利用される [5][62]。Captive Portal では、外部のサーバへの通信は横取りされ、通信を開始する前に利用者に対して認証を要求する。一般にこの Captive Portal は、ルータにおいて、IP アドレスのレベルでアクセス制御を行う。Suzuki らの方法 [57] でも、同様に外向きアクセス制御を行うが、この方法では、ルータにおいて DNS のレベルでアクセス制御を行うことができる。これらの方法ではどれも、URL レベル外向きアクセス制御を行うことができない。本研究の手法では、Web プロクシにおいてアクセス制御を行うため、アプリケーション層で利用可能な情報、たとえば宛先 URL や HTTP メソッドなどの情報を用いてアクセス制御を行うことができる。

4.1.4 Web プロクシ認証と SPNEGO

RFC 2616 [20], RFC 2617 [23] では、Web ブラウザと Web プロクシサーバの間でユーザ認証を行う方法を定めている。これらの方法では、Web プロクシサーバは Web ブラウザに対し、HTTP レスポンス内の `Proxy-Authenticate` ヘッダを用いて認証情報を送るように要求する。その場合、Web ブラウザは Web プロクシサーバに対し、HTTP リクエスト内の `Proxy-Authorization` ヘッダに認証情報を格納し、Web プロクシサーバに提示する。これらの方法では、認証方法として Basic 認証と Digest 認証を利用することができる。

SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism)[65] は、Web ブラウザと Web サーバの間で認証方法を調整しながら認証処理を行うことができる認証システムである。SPNEGO に対応したシステムでは、GSS-API (Generic Security Service Application Program Interface)[42] に対応した認証方法を利用することができる。そのような認証方法として、Kerberos[52] や NTLM(Windows NT LAN Manager)[25] がある。

この Web ブラウザと Web プロクシサーバの間で認証を行う方法と SPNEGO は、併用することができる。たとえば、`Proxy-Authorization` ヘッダと SPNEGO による認証を併用することができる[1]。しかし、Web プロクシサーバにおいて SPNEGO を用いることには、いくつかの問題がある。その問題については、4.2 節で詳しく述べる。

[1] http://sourceforge.net/projects/squidkerbauth/
http://wiki.squid-cache.org/Features/NegotiateAuthentication

4.1.5 Shibboleth 化された Web Proxy

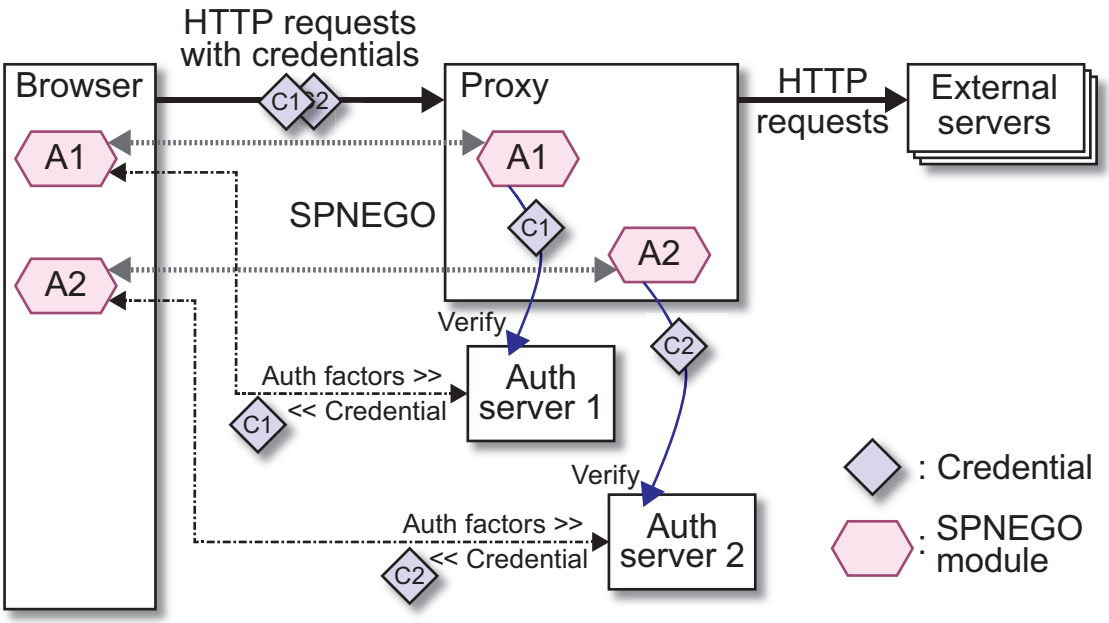
Komura ら [36] は、Shibboleth 認証を Web プロクシサーバにおいて利用可能とし、URL レベル外向きアクセス制御を実現している。この方法では、Web プロクシサーバは利用者をクッキー [39] を用いて識別している。Web ブラウザが Web プロクシサーバに対して外部の Web ページを要求すると、そのリクエストは、プロクシサーバと同じサーバ上で動作する Shibboleth の認証サーバへ転送される。この Shibboleth 認証サーバにおいて、ユーザ属性に基づいてアクセス可否が判断され、許可される場合には **phantom URL** と呼ばれる特別な URL へさらにリダイレクトされる。この phantom URL は、そのドメインとして外部 Web ページと同じドメインを持つ。この phantom URL において Web プロクシサーバはセッションクッキーをリクエストに挿入し、Web ブラウザに送信する。その後、最初にアクセスしようとしていた外部の Web ページへリダイレクトする。Web ブラウザは再びそのページへアクセスするが、その際、phantom URL により挿入されたセッションクッキーとともにリクエストを送信する。Web プロクシサーバはそのセッションクッキーの正当性を確認し、アクセスを許可する。この phantom URL によるクッキー挿入処理は、宛先となる外部 Web ページのドメインごとに行われる必要がある。

この Komura らの方法には、3 つの問題がある。1 つ目は、特別な URL(phantom URL) を使ったり複雑にリダイレクトを繰り返したりしなければならない点である。2 つ目は、利用者はアクセス先ドメインが変わるたびに、phantom URL とリダイレクトを使ったクッキー挿入処理を行わなければならない点である。3 つ目は、Shibboleth に特化した仕組みである点である。これに対し本研究で提案する手法では、認証処理はシンプルであり、また、アクセス先ドメインが変わるたびに何らかの処理を必要とすることはない。加えて、様々な認証方法に対して汎用的に対応することができる。

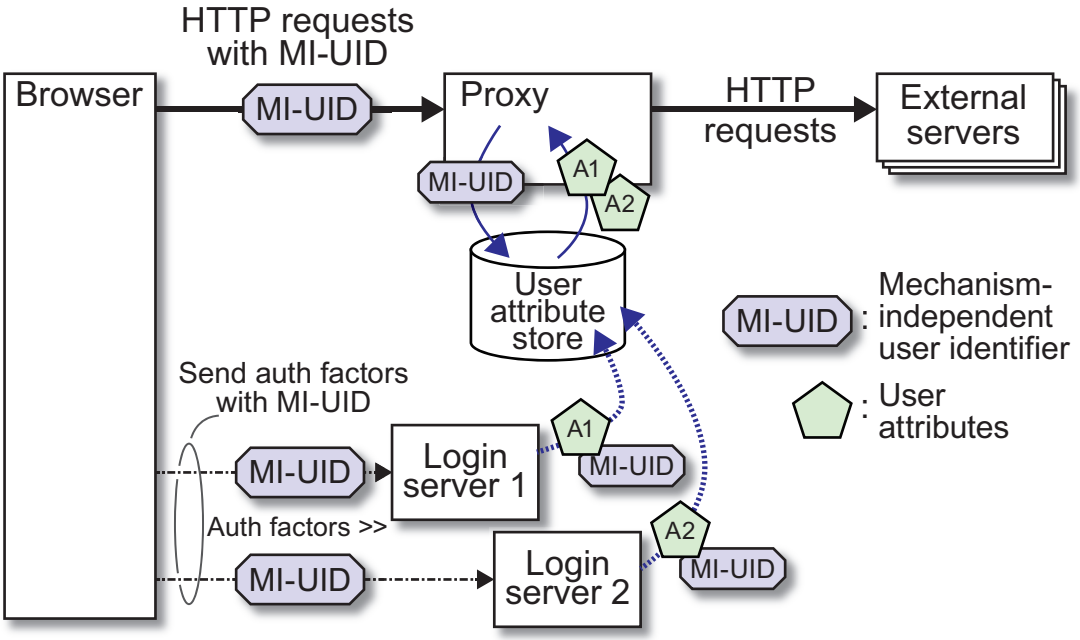
4.2 多数の認証システムを利用可能な URL レベル外向きアクセス制御

図 4.1 に、SPNEGO による従来の Web プロクシによる認証システムの認証処理の流れと、本研究で提案するアクセス制御システムの認証処理の流れを示す。図 4.1(a) に示すように、2 種類の認証システムに対応した SPNEGO による従来の Web プロクシによる認証システムは、Web ブラウザ、Web プロクシサーバ、そして 2 つの認証サーバから構成される。まず、Web ブラウザは、認証因子を認証サーバに送る。この認証因子は、通常はユーザ名やパスワードを用いるが、PKI(Public Key Infrastructure) による認証の場合には、X.509 証明書を認証因子として送る。次に、認証サーバは送られてきた認証因子を用いてユーザの認証を行う。この認証処理に成功した場合には、認証サーバはクレデンシャルを Web ブラウザに返す。ここで、クレデンシャルとは、そのユーザが認証を受けたことを証明する情報を意味する。続いて、Web ブラウザは Web プロクシサーバに対して、アクセス先 URL とともに、このクレデンシャルを SPNEGO を使って送信する。最後に、Web プロクシサーバは送られてきたクレデンシャルを検証し、そのクレデンシャルとアクセス先 URL に基づいてアクセスの可否を決定する。

従来の SPNEGO を用いた方法は、認証方法として Kerberos を利用することができる。しかし、Shibboleth[32] や OAuth[18] のような、複雑な認証システムを利用しようとした場合、いくつかの問題がある。1 つ目の問題は、SPNEGO では、Web ブラウザと Web プロクシサーバは、認証システムに依存した方法によって通信を行うことである。そのため、新た



(a) Collaboration among a browser, authentication servers, and a proxy in SPNEGO.



(b) Collaboration among a browser, authentication servers, and a proxy in proposed method.

図 4.1: Web ブラウザと Web プロクシサーバによる認証の概要

な認証システムに対応するためには、その認証システムのためのプロトコルを定義しなければならない。Shibboleth 認証に対応するためには、利用者の属性情報を SAML(Security Assertion Markup Language)[46] 形式でやりとりする必要がある。OAuth 認証に対応するためには、アクセストークンをどのようにして Web ブラウザから Web プロクシサーバへ受け渡すかを定義する必要がある。

2 つ目の問題は、新たな認証システムに対応するためには、Web ブラウザ側と Web プロクシ側の両方に手を加える必要があることである。図 4.1(a) の Web ブラウザには、2 つの拡張が存在し、Web プロクシ側にも同様に 2 つの拡張が存在している。このような拡張を開発することは、言うまでもなく相応の努力が必要とされる。たとえば、Shibboleth 認証に対応するためには、そのための拡張機能を Firefox, Internet Explorer, Safari, Opera, Google Chrome といった様々な Web ブラウザのために開発しなければならず、また Web プロクシ側についても同様に開発を行わなければならない。SPNEGO のように HTTP ヘッダを利用して利用者を識別する場合にこのような問題が生じるが、その代わりにクッキーを用いて属性情報やアクセストークンを Web ブラウザから Web プロクシサーバへ受け渡すこともできる。しかし、クッキーを用いることには、4.3.1 項で述べるような問題がある。

本研究では、これらの問題を図 4.1(b) に示すような方法によって解決する。この Web プロクシによるアクセス制御システムは、Web ブラウザ、Web プロクシサーバ、2 つの**ログインサーバ**と**ユーザ属性ストア**から構成される。まず、Web ブラウザは認証因子をログインサーバに送る。この際、認証因子とともに**認証方法に依存しないユーザ識別子 (Mechanism-Independent User Identifier: MI-UID)** を送る。次に、ログインサーバは受け取った認証因子を用いてユーザ認証を行う。もしこの認証処理に成功した場合には、ログインサーバは認証結果として得られる属性情報を、MI-UID と紐付けてユーザ属性ストアへ格納する。続いて、Web ブラウザは Web プロクシサーバに対して、アクセス先の URL とともに、MI-UID を送信する。最後に、Web プロクシサーバは MI-UID を用いて属性情報をユーザ属性ストアから取り出し、その情報とアクセス先 URL に基づいてアクセスの可否を決定する。

本研究の手法を用いることにより、SPNEGO による従来の方法に比べ、以下の利点がある。まず、本研究の手法では、Web ブラウザから Web プロクシサーバに対して、ユーザの属性情報やアクセストークンをやりとりするために使われるプロトコルを定義する必要がない。これは、Web ブラウザと Web プロクシサーバにおいて、SPNEGO モジュールのようなものを実装する必要がないことを意味する。

次に、新たな認証システムに対応する場合でも、ログインサーバを実装し、システムに加えるだけでよい。このログインサーバは、その認証システムを利用する通常の Web アプリケーションとして実装することができる。そのため、広く公開されているライブラリやツールを使って、簡単に実装することが可能である。加えて、この方法では、Kerberos や LDAP といった、既存の Web サーバにおける認証システム用のモジュールを利用することができるため、既存の認証システムに対しても容易に対応することができる。この Web アプリケーションの具体的な実装例については、4.3.5 項で述べる。

さらに、本研究の手法では、Web プロクシサーバにおける属性情報の検証処理が不要となる。SPNEGO の場合、属性情報は Web ブラウザから送出されるため、悪意のあるユーザが偽造を行う可能性がある。そのため、Web プロクシサーバでは送られてきた情報の正当性を検証する必要がある。この検証処理は、認証システムに固有の処理が必要となることがある。本研究の手法の場合、属性情報は信頼されたログインサーバとユーザ属性ストア、Web プロクシサーバの間でのみやりとりされる。そのため、Web プロクシサーバにお

いて、信頼されたユーザ属性ストアから取得した属性情報を検証する必要がない。

最後に、本研究の手法では、一人のユーザが複数の認証システムを同時に利用して認証を受けることができる。たとえば、Facebook を使ってログインしているユーザが、さらに Shibboleth を使って追加でログインすることができる。この場合、Shibboleth でさらにログインしたことにより、このユーザは Facebook を用いた場合にアクセスが許可される範囲と、Shibboleth を用いた場合にアクセスが許可される範囲の両方に対して、アクセスを行うことができるようになる。

この本研究の手法には、MI-UID の実装上の問題がある。この識別子の要件については、次項で述べる。

4.2.1 認証方法に依存しないユーザ識別子: MI-UID

本研究において提案する、Web プロクシによるアクセス制御システムでは、認証方法に依存しないユーザ識別子である MI-UID を必要とする。これは、Web ブラウザ、Web プロクシサーバ、ログインサーバとユーザ属性ストアが協調して動作し、アクセス制御を実現する上で重要なものである。この識別子は、次に示す特徴を持つことが求められる。

- ローカルエリアネットワーク (LAN) において、ユニークなものであること
- 認証方法に依存しないものであること
- 認証処理の前後で変化しないものであること
- なりすましに対して耐性を持つこと
- Web ブラウザから Web プロクシサーバに対して、HTTP リクエストごとに受け渡されるものであること

4.2.2 ユーザ属性ストア

ユーザ属性ストアは、ユーザ識別子に紐付けられた属性情報を格納する、信頼された Key-value ストアである。このユーザ属性ストアは、Web ブラウザから隔離されている必要がある。

このユーザ属性ストアは、次のようなエントリを保持する。

```
(<user identifier>, [<attribute_1>, ...])
```

各エントリは、ユーザ識別子と、属性情報のリストから構成される。ユーザがログインすると、ログインサーバによりこのエントリが作成される。このエントリはユーザがログアウトすると削除される。

このユーザ属性ストアは、次のような手続きを提供する。

`append_attr(uid, attr)`

この手続きでは、`uid` で示されるユーザのエントリに、`attr` で示される属性を追加する。もしエントリが存在しない場合には、エントリが新たに作成される。この手続きは、ログインサーバにより実行される。

`get_attr(uid)`
: この手続きは、uid で示されるユーザに紐付けられた属性情報のリストを返す。この手続きは、Web プロクシサーバにより実行される。

`remove_attr(uid)`
: この手続きでは、uid で示されるユーザのエントリを削除する。この手続きは、ログインサーバにより実行される。

4.2.3 Web プロクシサーバ

Web プロクシサーバでは、URL レベルでの外向きアクセス制御を行う。この Web プロクシサーバは、LAN と外部ネットワークの境界に設置され、LAN 内にある Web ブラウザからの HTTP リクエストを外部のネットワークにある Web サーバへ中継する。ネットワーク管理者はアクセス制御のポリシを記述し、この Web プロクシサーバ上に設置する。

本研究の手法では、ネットワーク管理者は認証システムにより提供される属性情報を用いてポリシを記述することができる。たとえば、図書館のネットワーク管理者は、そのユーザの大学における所属情報が教員である場合には外部のあらゆる Web ページへのアクセスを許可し、学生である場合には電子ジャーナルへのアクセスのみを許可する、といったポリシを記述することができる。

管理者は、このポリシを認証方法に依存しない形で記述することができる。もし新たな認証システムに対応する場合でも、その認証システムのためのポリシ記述を可能とするために新たに Web プロクシサーバに拡張機能を追加する必要はない。たとえば、ユーザが何らかの認証システムから E メールアドレスを提供すれば、外部の Web ページを閲覧することを認める、といった記述を行うことができる。この E メールアドレスという属性情報は、OAuth や Shibboleth、または他の認証システムから得られたものを使うことができる。

4.2.4 ログインサーバ

ログインサーバは、ユーザ認証処理を実行し、ユーザの属性情報をユーザ属性ストアに格納する。認証システムにおいては、1 回の認証処理において、複数の認証サーバを相手にやりとりを行う必要があるものが存在する。たとえば、Shibboleth の場合、Shibboleth Service Provider (SP) と、Shibboleth Identiry Provider (IdP) の 2 つのサーバを相手に、ユーザ認証処理が行われる。本研究の手法では、そのような場合においても適切に認証処理を行うことができる。

ログインサーバは、認証システム側の認証サーバと Web ブラウザの間で行われる認証処理については関与しない。これによって、ログインサーバのモジュール性を高めている。このログインサーバでは、対象とする認証システムに固有の方法により認証処理を行い、ユーザの属性情報を取得する。Shibboleth の例では、Shibboleth 用のログインサーバは、Shibboleth IdP との間で Security Assertion Markup Language (SAML) [46] を用いてトークンのやりとりや属性情報の取得を行う。Facebook の例では、Facebook 用のログインサーバは、Facebook 側の認証サーバとの間でトークンを用いた遠隔手続き呼び出しを行い、属性情報の取得を行う。このようにしてユーザの属性情報を取得したあと、ログインサーバはそれらをユーザ属性ストアへ格納する。Web プロクシサーバは、その属性情報を、統一的な方法によりユーザ属性ストアから取得することができる。

Shibboleth と Facebook のログインサーバの実装については、4.3.5 項で述べる。

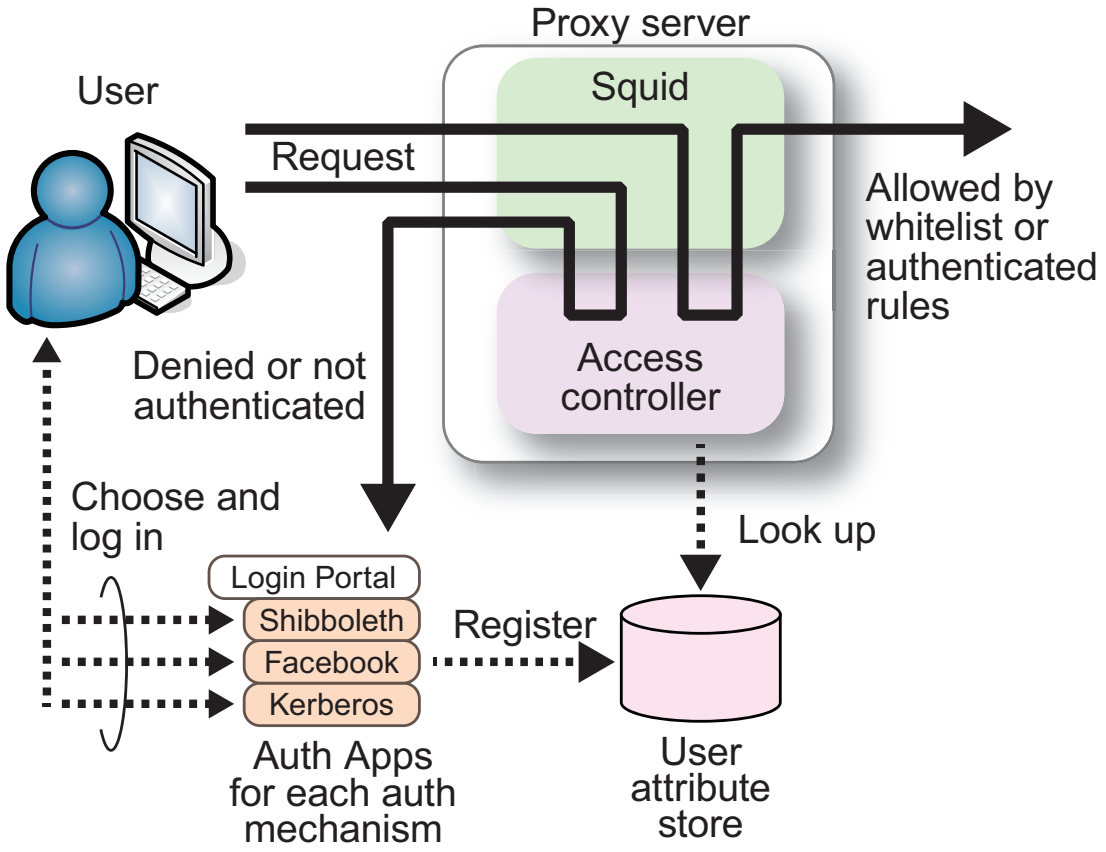


図 4.2: URL レベル外向きアクセス制御システムの実装の概要

4.3 実装

4.2 節では、Web プロクシサーバを用いた URL レベル外向きアクセス制御について述べた。本節では、その実装について述べる。この実装では、Shibboleth 認証や OAuth による Facebook 認証、そして既存の認証システムの例として Kerberos 認証に対応する。

図 4.2 に、実装したシステムの概要を示す。このシステムは Web ブラウザ、Web プロクシサーバ、ユーザ属性ストア、**ログインポータル**、そして **Auth App** からなる。このうち、Web プロクシサーバは、Squid プロクシサーバ部分と、**アクセスコントローラ**と呼ぶ拡張モジュールからなる。Squid プロクシサーバは、HTTP リクエストを受け取るたびにこのアクセスコントローラを呼び出す。アクセスコントローラは、そのリクエストの内容と管理者により記述されたポリシに応じて、アクセスを許可するか、拒否するかを決定する。

ログインポータルは、ユーザが認証システムを選択するためのポータルページである。このページでは、ユーザは外部の Web ページを閲覧するために必要な認証システムを選択することができる。その認証システムに固有の認証処理は、Web アプリケーションとして実装されている Auth App によって行われる。

4.3.1 MI-UID の実装

4.2.1 項において、認証方法に依存しないユーザ識別子である MI-UID の要件について述べた。本項では、そのようなユーザ識別子の 3 種類の実装方法について述べる。

まず 1 つ目の実装方法は、HTTP ヘッダを利用するものである。MI-UID を実装するために、次のような 2 つの新たな HTTP ヘッダを提案する。

```
Proxy-Set-Cookie: <set-cookie-string>
Proxy-Cookie: <cookie-string>
```

これらのヘッダは、Web プロクシサーバが、Web プロクシサーバと Web ブラウザの間における認証のためのクッキーを扱うことができるようにするものである。1 つ目のヘッダは RFC2109[39] における `Set-Cookie` ヘッダに似ているが、Web プロクシサーバから Web ブラウザに対しての応答の際に用いられるという点で異なる。2 つ目のヘッダは `Cookie` ヘッダに似ているが、Web ブラウザがユーザ識別子をログインサーバと Web プロクシサーバに提示するために用いられるという点で異なる。

2 つ目の実装方法は、同じく HTTP ヘッダを利用するものであるが、新たに HTTP ヘッダを提案する代わりに、既存のヘッダを用いるものである。

```
Proxy-Authorization: Basic <rand_uid>:<rand_password>
```

このヘッダは、RFC2617[23] における `Proxy-Authorization: Basic` ヘッダに似ているが、ユーザを識別するためにランダムな UID とランダムなパスワードを用いる点で異なる。あるログインセッションにおいて、Web ブラウザは同じ UID とパスワードを送り続ける必要がある。

これらの 2 つの実装方法は、既存の Web ブラウザや Web プロクシサーバに対する改変が必要となる。

3 つ目の実装方法は、HTTP ヘッダを用いないものである。認証方法に依存しないユーザ識別子として、Web ブラウザを実行している計算機の IP アドレスを用いる。この IP アドレスは、4.2.1 項で述べた要件を満たしている。計算機の IP アドレスは LAN 内でユニークであり、また認証方法に依存しない。計算機の IP アドレスはユーザのログイン中に変化しない。また、インテリジェントスイッチ[2]を用いることにより、計算機の IP アドレスの詐称を防ぐことができる。また、ユーザ名とパスワードを利用してアクセスを制御するようないくつかのルータには、ソース IP アドレスを用いて利用者の識別を行うものがある [5][62]。本研究で対象とするネットワークにおいては、IP アドレスは MI-UID としての要件を満たしていることから、本実装においては、この IP アドレスを MI-UID として用いる。

IP アドレスを用いることにより、システムの実装をシンプルにすることができる。IP アドレスを用いる上で、Web ブラウザに変更を加える必要はない。また、Web ブラウザを実行している計算機の IP アドレスは、Web プロクシサーバやログインサーバにおいて、容易に取得することができる。

その一方で、IP アドレスを用いる場合、いくつかの制約が生じる。まず、IP アドレスを共有する仕組みを使うことができなくなる。たとえば、LAN 内において NAT(Network Address Translation) を使うことができない。次に、インテリジェントスイッチの機能などを用いて、IP アドレスの詐称を防ぐという措置を執る必要がある。加えて、IPv6 におけるステートレスアドレス自動設定のプライバシ拡張 [45] のような機能を使うことができない。IP アドレスによるトラッキングを防ぐ場合には、この機能をクライアントではなく Web プロクシサーバにおいて有効にする必要がある。

[2]Cisco Catalyst 2960 スイッチは Dynamic Host Configuration Protocol (DHCP) スヌーピング機能、動的 Address Resolution Protocol (ARP) インスペクション機能、IP ソースガード機能を持ち、これらの機能を用いることで IP アドレスの詐称を防ぐことができる。

4.3.2 ユーザ属性ストア

ユーザ属性ストアは、ユーザ識別子に紐付けられた属性情報を格納する、信頼された Key-value ストアである。本研究では、これを協調言語 Linda[24] のタプルスペースを用いて実装した。開発言語として Ruby を用いることから、Ruby 用のタプルスペースとして、Rinda[50] を用いた。

タプルスペースの例は次のようになる。

```
[ MAGIC_attr, uid1, attr1_shib  ]
[ MAGIC_attr, uid1, attr1_oauth ]
[ MAGIC_attr, uid2, attr2_shib  ]
[ MAGIC_attr, uid3, attr3_oauth ]
```

最初の要素 MAGIC_attr は、このタプルが属性情報を格納していることを示す識別用の文字列である。2 番目の要素はユーザ識別子で、3 番目の要素は属性情報である。この例では、ユーザ識別子 uid1 で表されるユーザは、attr1_shib と attr1_oauth の 2 つの属性情報を持っている。

タプルは、次のようにしてログインサーバにより作成される。

```
space.write( [MAGIC_attr, uid, attr] )
```

write() 手続きにより、ブラケットで囲まれた内容のタプルがタプルスペース上に作成される。ログインサーバは、ユーザ認証処理を行ったあと、この手続きを呼び出してタプルを作成する。複数のログインサーバがある場合、一人のユーザに対して複数のタプルが作成される。

タプルは、次のようにして Web プロクシサーバにより読み出される。

```
attr_list = space.read_all( [MAGIC_attr, uid, nil] )
attr_list.each {|tuple|
   attr = tuple[2]
   use( attr )
}
```

read_all() 手続きにより、引数で渡されたパターンにマッチするタプルのリストを取得することができる。このコードでは、nil はワイルドカードを意味する。

4.2.2 項で述べたように、このタプルスペースサーバは Web ブラウザから隔離されている必要がある。本実装では、タプルスペースサーバ、Web プロクシサーバ、ログインポータルと Auth App を同一のホスト上で稼動させる。その上で、タプルスペースサーバはこれらのサーバからの接続のみを受け付けるようにしている。また、タプルスペースサーバは SSL の利用をサポートするため、SSL 証明書を用いてこれを保護することもできる。

4.3.3 Web プロクシサーバとアクセスコントローラ

Web プロクシサーバの実装には、オープンソースの Web プロクシサーバソフトウェアである Squid[4] を利用した。Squid では、HTTP リクエストの宛先 URL を書き換えるための拡張機能が提供されている。この拡張により呼び出されるプログラムは、**URL rewrite**

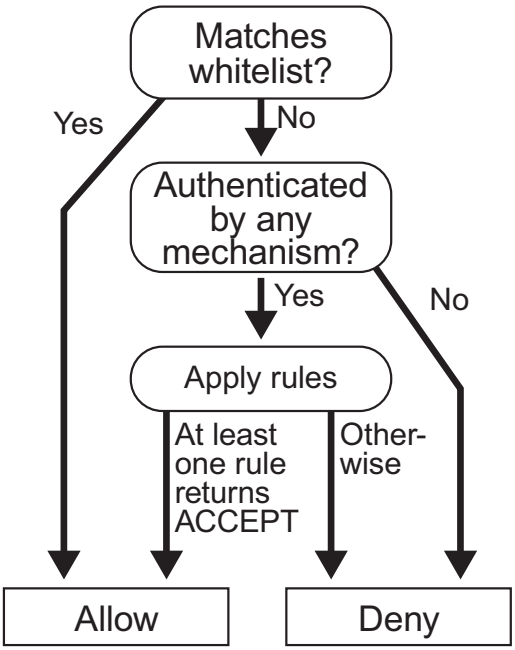


図 4.3: アクセスの可否を決定する流れ

program と呼ばれる。今回はアクセスコントローラを、この URL rewrite program として実装した。

Squid が起動されると、アクセスコントローラのプロセスが生成される。Squid が Web ブラウザから送られる HTTP リクエストを受け付けるたびに、その URL とクライアントの IP アドレス、HTTP リクエストメソッドの情報が、パイプを経由してアクセスコントローラのプロセスへ渡される。アクセスコントローラでは、そのリクエストを許可する場合には元のアクセス先 URL を返す。Squid はその URL に対してリクエストを中継し、その応答をクライアントに中継する。リクエストを拒否する場合には、ログインポータルの URL を返す。Squid はクライアントに対し、このログインポータルへのリダイレクトを行う。

アクセスコントローラは Squid から URL、IP アドレス、リクエストメソッドの情報を受け取ると、図 4.3 に示すような処理によってアクセスの可否を決定する。まず、アクセスコントローラは、アクセス先 URL が管理者により記述されたホワイトリストにマッチするかを確認する。もしマッチした場合には、アクセスコントローラはアクセス先と同じ URL を返し、そのリクエストは許可される。そうでない場合、次にユーザ属性ストアからそのユーザの属性情報を取得する。もし属性情報が取得できなかった場合には、それはそのユーザが認証を受けていないことを意味するため、アクセスコントローラはログインポータルの URL を返し、その URL へリダイレクトさせる。

ユーザ属性ストアから属性情報が取得できた場合、アクセスコントローラはあらかじめ定義されたルールとのマッチングを行う。このルールの詳細については、4.3.4 項で詳しく述べる。アクセスを許可するルールが 1 つでもあれば、アクセスコントローラはアクセス先と同じ URL を返し、そのリクエストは許可される。そうでない場合には、リクエストは許可されず、同様にログインポータルへリダイレクトさせる。

アクセスコントローラでは、性能向上のために属性情報をキャッシュする仕組みを持つ。その一貫性を保つため、本実装では Rinda の更新通知機能を用いている。

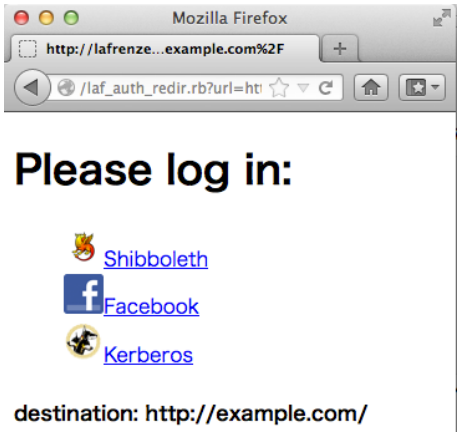


図 4.4: ログインポータル

4.3.4 アクセス制御ルールの記述

ネットワーク管理者は、アクセス制御ポリシをルールのリストとして記述し、アクセスコントローラに登録する。各ルールは、条件部と本体からなる。条件部は、どのようなユーザを対象とするかを記述し、本体は URL のパターンとマッチした際のアクションのリストを記述する。

アクセスコントローラでは、ルールを次のように評価する。まず、ユーザの属性情報を使って条件部を評価する。もし条件を満たしていれば、本体を評価する。本体は URL のパターン、リクエストメソッドと、マッチ時のアクションのリストからなる。URL パターンは正規表現で表され、アクションは ACCEPT または REJECT を指定する。本実装では、ルールは YAML 形式で記述する。

ルールの例は次のようになる。

```
- cond:
    affiliation: "student@.+"
  do:
    - url:      "http://example\.com"
      action:   REJECT
    - url:      "http://bbs\.example\.net"
      method:   POST
      action:   REJECT
  default_policy: ACCEPT
```

このルールは、affiliation 属性が “student” であるユーザを対象としたものである。このユーザが example.com へのリクエストを行った場合、それは拒否される。このユーザが bbs.example.net に対し、HTTP POST リクエストを行った場合にも、それは拒否される。そうでない場合、デフォルトポリシが ACCEPT に設定されているため、リクエストは許可される。

4.3.5 ログインポータルと Auth App

ログインポータルは、ユーザがどの認証システムを利用するかを選択することができるポータルページである。図 4.4 にログインポータルのスクリーンショットを示す。このペー

```
#!/usr/bin/env ruby
require 'rinda/tuplespace'

uri   = "http://attr-store-host:port/"
space = DRbObject.new_with_uri(uri)
uid   = ENV["REMOTE_ADDR"];
attrs = {"mechanism"   => "shibboleth",
         "email"       => ENV['mail'],
         "affiliation" => ENV['affiliation'],
         "persist_id"  => ENV['persistent_id']}
space.write(["MAGIC_attr",uid,attrs ])
```

図 4.5: Shibboleth 用 Auth App の主要部分

```
AuthType shibboleth
AuthName "Shibboleth User Only"
ShibRequireSession On
ShibUseHeaders On
require valid-user
```

図 4.6: Shibboleth 用 Auth App で用いる .htaccess ファイル

ジでは、ユーザは Shibboleth, Facebook, Kerberos の中から認証システムを選択することができる。ページ中のリンクをクリックすることにより、その認証システムの Auth App へ移動することができる。

これまでに述べたように、Auth App は認証システムの認証サーバとユーザ属性ストアの仲介を行う Web アプリケーションである。Auth App は認証方法に固有の処理を行い属性情報を取得し、それをユーザ属性ストアに格納する。

Shibboleth 用 Auth App

図 4.5 に、Shibboleth 用の Auth App の主要部分を示す。これは Apache HTTP サーバ上で動作する CGI (Common Gateway Interface) プログラムであり、図 4.6 に示すような.htaccess ファイルにより、Shibboleth 認証で保護されている。このアクセス制御記述は、Apache の `mod_shib` モジュール [64] を有効化し、Shibboleth の認証を要求し、また認証後に Shibboleth のユーザ属性情報を環境変数経由で利用可能としている。

Web ブラウザが最初にこの Auth App にアクセスしたときには、Shibboleth の認証サーバ (IdP) へリダイレクトされる。このリダイレクトは、図 4.6 の記述により有効化された、`mod_shib` の機能により行われる。Shibboleth の IdP では、ユーザは自身のユーザ名とパスワードを入力する。IdP はこれを検証し、認証に成功した場合には、Auth App へリダイレクトさせる。再びこの Auth App にアクセスすると、IdP による認証を済ませたことから図 4.5 の処理が行われる。Auth App のプログラムでは、環境変数を経由して Shibboleth のユーザ属性情報にアクセスすることができる。ここでは、`affiliation` 属性 ("staff" や "student" が格納される) と `persistent_id` 属性 (匿名化されたユーザ識別子) の 2 つの属性情報を環境変数から取得している。これらの属性はハッシュテーブルに格納され、Auth

App はこれをユーザ属性ストアに格納する。この例における Shibboleth のための Auth App は、Ruby を用いて 14 行で記述されている。

Facebook 用 Auth App

図 4.7 に、Facebook 用の Auth App の主要部分を示す。これは Facebook の Web アプリケーションとして実装される。Shibboleth の場合 (図 4.5) と似ているが、次の 2 点で異なっている。1 点目は、Facebook の場合、ユーザが認証を受ける前か受けた後かを Auth App 側で判定する必要があることである。2 点目は、Shibboleth の場合は属性情報を環境変数から取得したが、Facebook の場合は Facebook OAuth ライブラリの `FacebookOAuth::Client` インスタンスから取得することである。

Facebook 用の Auth App では、OAuth のトークンが “code” と呼ばれるパラメータに格納される。もしこのトークンが与えられずに Auth App が呼び出された場合には、認証が行われていないことを意味するため、Auth App は Facebook の認証サーバへリダイレクトする。ユーザがそこで認証処理を行うと、再び Auth App へリダイレクトされるが、その際には “code” パラメータとともにアクセスが行われる。アクセストークンがこのパラメータにより与えられたときには、そのトークンを使って属性情報を取得する。図 4.7 では、Facebook の認証サーバから、Facebook 上のユーザ ID、ユーザの E メールアドレス、そしてそのユーザが所属するグループの情報を取得している。これらの情報はハッシュテーブルに格納され、Auth App はこれをユーザ属性ストアに格納する。このコードにおいて、`group` という属性として、`important_visitors` という値を設定している。これは認証システムに依存しない方法でグループを指定するものである。認証システムにより得られるグループの情報は時としてその識別子が多様となるため、管理者は Auth App において、認証システムごとにその名寄せの処理を記述することができる。

この `group` を用いるルールは次のように記述することができる。

```
- cond:
    group:     "important_visitor"
    email:     ".+@.+$"
  do:
    - url:     "http://.*example\.org/"
      action:   ACCEPT
  default_policy: REJECT
```

この例では、グループとして “important_visitor” に所属しており、E メールアドレスを管理者に提供している場合、`example.org` へのアクセスを許可している。

この例における Facebook のための Auth App は、Ruby を用いて 33 行で記述されている。

Kerberos 用 Auth App

本研究のアクセス制御システムでは、Web サーバで用いる認証システムも利用することができる。図 4.8 に、Kerberos 認証を利用するための Auth App の主要部分を示す。これは Apache HTTP サーバ上で動作する CGI プログラムであり、図 4.9 に示すような.htaccess ファイルにより、Kerberos 認証で保護されている。これは Shibboleth の場合のもの (図 4.5,

```
#!/usr/bin/env ruby
require 'cgi'
require 'rinda/tuplespace'
require 'oauth'
require 'facebook_oauth'

uri   = "http://attr-store-host:port/"
space = DRbObject.new_with_uri(uri)
uid   = ENV["REMOTE_ADDR"];

if cgi.params["code"].empty?
  mode = :redir_for_auth
else
  mode  = :auth_done
  token = cgi.params["code"][0]
end

client = FacebookOAuth::Client.new(
          :application_id     => APP_ID,
          :application_secret => APP_SECRET,
          :callback           => callback_url)

if mode == :redir_for_auth
  auth_url = client.authorize_url( :scope => 'user_groups,email' )
  print cgi.header(:location => auth_url)
elsif mode == :auth_done
  client.authorize(:code => token)
  fbgroups = get_fbgroup(client)
  group = fbgroups.includes? IMPORTANT_VISITORS :
           "important_visitors":""
  attrs = { "mechanism" => "facebook",
            "id"    => client.info["id"],
            "email" => client.info["email"],
            "group" => group}
  space.write(["MAGIC_attr",uid,attrs ])
end
```

図 4.7: Facebook 用 Auth App の主要部分

```
#!/usr/bin/env ruby
require 'rinda/tuplespace'

uri   = "http://attr-store-host:port/"
space = DRbObject.new_with_uri(uri)
uid   = ENV["REMOTE_ADDR"];
princ = ENV["REMOTE_USER"];
princ_name = princ.split("@")[0]
email = "#{princ_name}@example.edu"
attrs = {"mechanism" =>"kerberos",
         "email"     => email,
         "principal" => princ}
space.write(["MAGIC_attr",uid,attrs])
```

図 4.8: Kerberos 用 Auth App の主要部分

```
AuthType Kerberos
AuthName "Kerberos User Only"
KrbMethodNegotiate On
KrbMethodK5Passwd Off
KrbAuthRealms REALM.EXAMPLE.EDU
Krb5KeyTab /etc/httpd/conf/keytab
require valid-user
```

図 4.9: Kerberos 用 Auth App で用いる .htaccess ファイル

図 4.6) と似ている。図 4.9 のアクセス制御記述は、mod_auth_kerb[3] を有効化し、Kerberos 認証の設定を行っている。図 4.8 のプログラムでは、Kerberos のプリンシパル名を環境変数から取得し、その値をもとに E メールアドレスの文字列を作成している。その後、この E メールアドレスとプリンシパル名をハッシュテーブルに格納し、ユーザ属性ストアに格納する。

このようにして、シンプルなパスワードファイルによる認証や、LDAP(Lightweight Directory Access Protocol)、SSL クライアント認証といった、既存の認証システムについても対応可能とすることができる。

4.4 評価

本節では、本実験で提案するアクセス制御システムの評価について述べる。

4.4.1 筑波大学 附属図書館への適用

本研究で実装したアクセス制御システムを、筑波大学 附属図書館において、182 台のキオスク端末を対象として適用した。キオスク端末では、来館者は OPAC サービスの Web

[3] http://modauthkerb.sourceforge.net/

ページ、図書館の情報を掲載したページ、バスの時刻表サービス、電子ジャーナルなどに対し、認証なしでアクセスすることができる。大学の学生や教員は、Shibboleth 認証を行うことにより、それ以外の外部の Web ページを閲覧することができる。

このシステムは、キオスク端末の Web ブラウザ、Web プロクシサーバ、ユーザ属性ストアと Shibboleth のログインサーバから構成される。このうち、キオスク端末の Web ブラウザでは、Web プロクシサーバの設定が行われており、このブラウザからの HTTP リクエストは、このシステムの Web プロクシサーバを経由する。ログインサーバは、学内に設置された Shibboleth の認証サーバに接続されている。これにより、e ラーニングシステムの Moodle[17] などの他のサービスにおいても、Shibboleth のシングルサインオンの恩恵を得ることができる。

この運用において、Web プロクシサーバ、タプルスペースサーバ、ユーザ属性ストアとログインサーバはどれも単一の仮想計算機上で動作している。仮想計算機のホストは、CPU として Intel Xeon X5570 を持ち、1000BASE-T の NIC を備える。また、サーバとして動作する仮想計算機には、2GB のメモリが割り当てられており、OS として CentOS 5.8 を稼動させている。全てのキオスク端末は、このサーバと 1000BASE-T で接続されている。

このアクセス制御システムを、2011 年 5 月から現在に至るまで、附属図書館において運用している。このシステムでは、Shibboleth 認証のみを利用できる設定となっている。本研究では、運用上得られたログの解析を行った。運用期間において、最もリクエストが集中した日である 2012 年 1 月 12 日のアクセスログを図 4.10 に示す。X 軸は時間を、Y 軸は 1 秒あたりのアクセス数を表している。

サーバを稼働させる計算機の性能は、182 台のキオスク端末からのリクエストを処理するだけの十分な性能を持っている。最もリクエストが集中した場合でも、その頻度は 381 リクエスト毎秒であった。この日の全てのリクエストは 587,325 件あり、そのうち 76%が認証を伴ったアクセスであった。残りのリクエストは、ホワイトリストにマッチしたことにより許可されたアクセスであった。また、この日に行われたユーザ認証処理はのべ 575 回であった。ホワイトリストは 712 個のエントリがあり、これは図書館のネットワーク管理者により管理されている。2011 年 5 月から 2012 年 4 月までの期間において、約 8,800 人のユーザがこのシステムを利用している。

4.4.2 新たな認証システムに対応する場合の容易さ

本研究の手法では、新たな認証システムが登場した場合でも、Web アプリケーションからその認証システムを利用可能である限り、容易に対応することができる。

従来の SPNEGO の場合では、新たな認証システムに対応する場合、その認証システムで取り扱うことができる属性情報や必要となるトークンを受け渡す方法をプロトコルとして定義し、その認証システムに固有な方法で認証処理を行う拡張モジュールを作成し、Web ブラウザと Web プロクシサーバの両者にインストールした上で、Web プロクシサーバ上にポリシを記述する必要があった。本研究の手法では、MI-UID によりプロトコルの定義やモジュールのインストールが簡素化される。認証システムに固有な方法で認証処理を行うログインサーバを作成し、Web プロクシサーバ上にポリシを記述するだけでよい。

また、このログインサーバの実装に関しても、4.3.5 項で示したとおり、Shibboleth の例では 14 行、Facebook の例では 33 行と、容易に記述することができる。

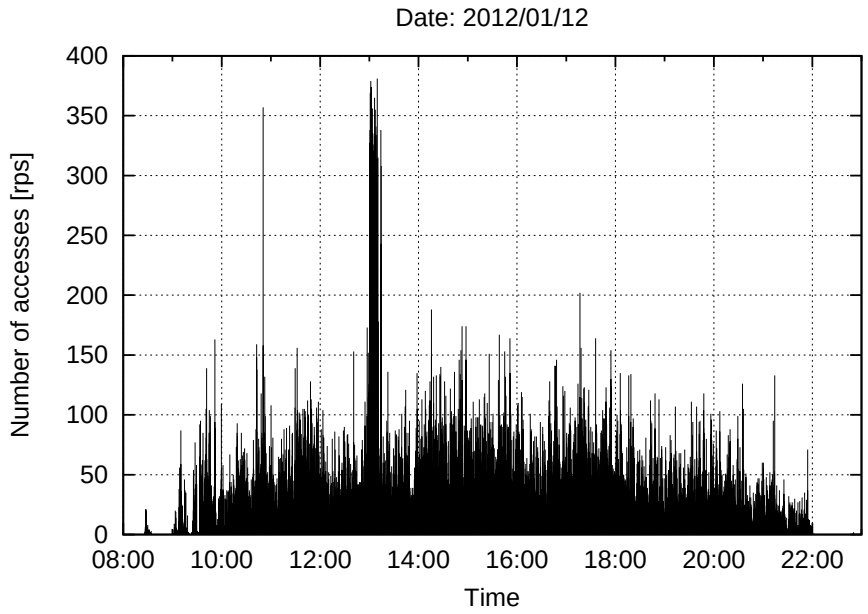


図 4.10: 最もリクエストが集中した日 (2012 年 1 月 12 日) のアクセスログ

4.4.3 スケーラビリティ

本研究の手法では、ユーザの属性情報を Web プロクシサーバとログインサーバの間で共有するために、ユーザ属性ストアという信頼された Key-value ストアを利用している。この手法では、クライアントからのリクエスト数の増加が起こった場合でも、ログインサーバや Web プロクシサーバを複数設置することにより、対応することができる。

このシステムにおけるスケーラビリティは、ユーザ属性ストアに依存する。本実装では、Ruby におけるタプルスペースの実装である Rinda を用いている。ユーザ属性ストアにおける、属性情報のキャッシュ一貫性を維持するため、Rinda の更新通知機能を用いている。本実装では、4.4.1 項で述べたように、少なくとも 381 リクエスト毎秒の要求を 1 台の計算機で処理することができる。

より高い性能を実現するためには、Rinda よりも高速なアクセスが可能な Key-value ストアを用いてユーザ属性ストアを実装すればよい。そのような Key-value ストアとして、memcached[16] がある。比較実験環境として、4.4.1 項で述べた計算機とは異なる計算機を用いて、簡単なベンチマークを行った。図書館において得られたログから、実際の運用の際に書き込まれたものと同等の内容をユーザ属性ストアに 10,000 件書き込み、その処理速度を測定した。Rinda による実装の場合、平均 2,882[write/s] であった。memcached による実装の場合、平均 11,597[write/s] であった。このユーザ属性ストアの書き込み処理は、ユーザのログイン処理の際に行われるものである。現実には、ユーザのログイン処理はそう頻繁には行われないため、1 台の計算機で処理する上で十分な処理性能を実現することができるといえる。もしより高い性能が要求される場合には、広く知られている memcached を用いたシステムにおけるスケーラビリティ向上のための手法を適用することができる。

4.5 まとめ

本章では、集中管理された利用者用計算機のうち、図書館における蔵書検索用の端末において、多数の認証システムを利用可能な URL レベル外向きアクセス制御について述べた。この仕組みを用いることにより、利用者は多数の認証システムの中から選択した認証システムを用いて認証を受け、Web アクセスを行うことを可能とした。この方法では、Shibboleth や OAuth をはじめとする多数の認証システムに対応することができる。また、この方法では、新たな認証システムに対応する場合でも、その認証システムのためのログインサーバを設置するだけでよい。このログインサーバが通常の Web アプリケーションとして記述できることにより、広く利用されているライブラリやツールを用いて、対応したい認証システムのためのログインサーバを容易に実装することができる。本研究では、そのようなログインサーバとして、Shibboleth 認証と Facebook の OAuth 認証のための実装を示した。これらは Apache HTTP サーバ上で動作するシンプルな CGI プログラムとして記述することができ、その主要部分のコード量は、Shibboleth では 14 行、Facebook では 33 行であった。従来の SPNEGO の場合と比べ、これらの認証システムに対応する上で、属性情報をやりとりするためのプロトコルを定義したり、Web ブラウザと Web プロクシサーバの拡張を実装したりする必要はない。

本研究の手法では、認証方法に依存しないユーザ識別子 (MI-UID) を用いて、信頼された Key-value ストアに対してログインサーバがユーザの属性情報を格納したり、Web プロクシサーバが属性情報を取り出したりする。Web ブラウザは Web プロクシサーバに対してリクエストを送る際に、この MI-UID を同時に送る。この MI-UID を実装する方法として、HTTP ヘッダを用いる 2 つの方法と、IP アドレスを用いる方法について述べた。

このようなアクセス制御システムを実装し、筑波大学附属図書館において運用を行っている。これは認証システムとして Shibboleth に対応し、182 台のキオスク端末からのリクエストを処理している。2011 年 5 月から運用を行い、一年間に約 8,800 人のユーザがこのシステムを利用している。

今後の課題として、MI-UID の他の実装方法として、ウェブバグ [27] などを用いてユーザを追跡する技術を利用することが挙げられる。たとえば、Web プロクシサーバにおいて HTML の内容を書き換え、ウェブバグを埋め込むことにより、利用者を識別することができると考えている。

第5章　結論

本論文では、多数の計算機と多数の利用者を持つ計算機システムにおける、集中管理された計算機システムの利便性を向上させる方法について述べた。

第 1 章では、本研究の背景である集中管理された計算機システムと、そのようなシステムにおける利便性について述べた。利用者が多くなるとその情報リテラシや計算機利用のスキルの分布も広がり、管理者は計算機システムを管理下に置くために情報リテラシや計算機利用のスキルが低い利用者にあわせたシステムを利用者に提供する。このことが情報リテラシや計算機利用のスキルが高い利用者の利便性を低下させてしまうことについて述べた。これを解決し、集中管理された計算機システムにおける利便性を向上させるため、利用者認証を強化することを提案した。利用者や使用する計算機などに応じて、利用者がより柔軟な利用を行うことができるような仕組みにより、幅広い層の利用者を持つ環境における集中管理された計算機システムにおいても、利便性を向上させることができることを述べた。集中管理された利用者用計算機の例として、大学における重要な計算機である (1) 計算機実習室に配置された計算機、(2) 図書館における蔵書検索用の端末 の 2 つを取り上げた。これらの利用者用計算機に対して利用者認証を強化することで計算機の利便性を向上させることが可能であることを示した。

第 2 章では、大学において集中管理されている計算機として、計算機実習室に配置された計算機を取り上げ、利用者認証を強化することにより利便性を向上させる手法を適用することについて述べた。通常よりも高い権限を持った OS を利用者に提供する場合、集中管理された利用者用計算機の管理者は、通常のユーザ名とパスワードによる認証では不十分であり、より厳密な方法で利用者や計算機を識別したいと考えることがある。本研究ではそのような要望に応えるため、利用者の識別に認証デバイスを、計算機の識別に TPM を用いる。これにより、管理者が認めた利用者と計算機の組み合わせでのみ、より高い権限を持った OS を起動可能とすることができる。また、その利用者がその計算機を利用していることを識別するために、従来の HTTPS クライアント認証では行うことができなかった、利用者と計算機の 2 種類の証明書を同時に使用した認証処理を行うための仕組みを提案・実装した。そのような OS の起動・終了制御システムを実装し、利用者と計算機の組み合わせに応じて OS の起動・終了制御を行うことができることを示した。また、利用者が操作する OS のデータの配信がボトルネックとなることについて述べた。

第 3 章では、この OS のデータの配信のボトルネックを P2P 方式のディスクイメージ配信システムにより解決することについて述べた。第 2 章で提案した OS の起動・終了制御の仕組みに限らず、一般的なネットワークブートシステムに共通して存在するこのボトルネックは、大学の実習室のような環境において問題を引き起こす。その問題とは、配信システムに起動処理が集中すると、クライアントの起動に長い時間がかかってしまうことである。本研究では、P2P 方式のディスクイメージ配信システムにより、この問題を解決した。先行研究として、P2P 方式のネットワークブートを行うシンクライアントシステム [66] がある。このシステムでは、KNOPPIX を配信対象とし、その起動処理に手を加えることによって効率的な配信を実現している。本研究では、大学の実習室のような環境を対象とす

るため、Windows のようなプロプライエタリな OS を含めて多数の OS を配信可能であることが求められる。また、実習室のような環境で運用を行う場合、同時に起動処理が開始されることはまれであり、実際には正規分布に従うような分布で起動処理が行われる。そこで本研究では、現実的な起動処理の分布として、正規分布に従って多数の計算機が起動処理を開始するような場合に、その起動所要時間がどのように変化するかについて明らかにすることを述べた。

Windows のような OS を含めて多数の OS を配信可能で、配信サーバからだけでなく既にそのデータを持っている他のクライアントからも取得を行うことにより、配信サーバの負荷やトラフィックを軽減する P2P 方式のディスクイメージ配信システムを設計・実装した。そのシステムを用いて、実習室においてみられる正規分布に従った起動処理において、その分布の幅 (時間幅) がどのような影響を与えるかを測定した。その結果、P2P 方式の場合、時間幅が大きくなればなるほど他のクライアントからデータを取得できる確率が上がり、配信サーバのトラフィックを軽減できることを示した。また、起動分布として最悪のケースである、全てのクライアントが同時に起動を開始する場合においても、従来の NFS の方法に比べ、P2P 方式の方が素早く起動を完了できることを示した。

第 4 章では、大学において集中管理されている計算機として、図書館における蔵書検索用の端末を取り上げ、利用者認証を強化することにより利便性を向上させる手法を適用することについて述べた。これは、蔵書検索端末で Web アクセスを行う際に必要とされる認証を、多数の認証システムを用いて行えるようにするものである。これにより、一時的アカウントの発行に伴う煩雑さを解消することを可能とした。このようなアクセス制御を実現する上で、認証方法に依存しないユーザ識別子とユーザ属性ストアを用いることにより、新たな認証システムに対応する場合でも、既存のライブラリやツールを用いて容易に対応することができることについて述べた。このような Web アクセス制御システムを設計・実装し、筑波大学 附属図書館において運用を行っている。認証システムとして Shibboleth に対応したこのシステムは、2011 年 5 月から現在に至るまで、182 台のキオスク端末を対象として運用されている。

このように、集中管理された利用者用計算機において、利用者認証を強化することによりその利便性を向上させることができることを、計算機実習室に配置された計算機と、図書館における蔵書検索用の端末において示した。

今後の課題としては、P2P 方式において配信サーバの帯域幅を広げた場合における最適化の検討やディスクイメージ配信システムにおけるオーバヘッドの除去、Web アクセス制御システムにおける認証方法に依存しないユーザ識別子として、ウェブバグを用いる方法の検討が挙げられる。

謝辞

本論文にまとめられている研究は、筆者が筑波大学大学院 博士課程 システム情報工学研究科 コンピュータサイエンス専攻に在籍中、新城靖先生、佐藤聡先生、板野肯三先生、中井央先生のご指導の下に行われたものである。

本論文をまとめる上で、板野先生、和田耕一先生、加藤和彦先生、新城先生、木村成伴先生には本研究の内容に関する的確なご指摘・ご指導を頂いた。ここに心より感謝を表す。

新城先生には、研究のあらゆる面において手厚いご指導を頂いただけでなく、研究に携わる人間としての姿勢、物事の捉え方、考え方など様々なご指導を頂いた。ここに心より感謝を表す。

佐藤先生には、研究の立ち上げから方針の議論、研究発表などの研究に関するご指導を頂いた。加えて、学生生活における様々な事柄やネットワークに関する最新動向などの議論にご協力いただいた。ここに心より感謝を表す。

本論文にまとめられている研究において、杉木章義先生には内容に関する的確なご指摘・ご指導を頂いた。ここに感謝を表す。

第 3 章で行った実験においては、中井先生をはじめ、筑波大学 学術情報メディアセンターの全学計算機システムに携わる方々のご協力を頂いた。ここに感謝を表す。

第 4 章で述べられている研究の一部は、2009 年度 システム開発型特別プロジェクト ICT ソリューション・アーキテクト育成プログラムとして行われた。この研究を進めるにあたり、齊藤剛氏、金子直矢氏にご協力を頂いた。ここに感謝を表す。また、筑波大学 附属図書館におけるアクセス制御システムの導入においては、筑波大学 附属図書館 研究開発室の徳田聖子氏、平田完氏、中山知士氏、真中孝行氏にご協力を頂いた。ここに感謝を表す。

ソフトウェア研究室の皆様には、本研究を進めるにあたり、研究の議論から日常の他愛のない雑談に至るまで、様々な面でご協力を頂いた。ここに感謝を表す。

最後に、筆者の学生生活を支えてくれた家族と友人、そして伴侶に、深く感謝を表す。

参考文献

[1] fusefs. http://rubyforge.org/projects/fusefs/.

[2] The messagepack project. http://msgpack.org/.

[3] rados block device. http://ceph.newdream.net/wiki/Rbd.

[4] squid : Optimising web delivery. *http://www.squid-cache.org/*.

[5] Guido Appenzeller, Mema Roussopoulos, and Mary Baker. User-friendly access control for public network ports. In *IEEE INFOCOM*, pp. 699–707, 1998.

[6] Apple Inc. Mac OS X Server: NetBoot Technology Brief. http://manuals.info.apple.com/en/NetBoot_TB_v10.4.pdf.

[7] Apple Inc. Mac OS X v10.5, 10.6: About the Parental Controls Internet content filter. http://support.apple.com/kb/HT2900.

[8] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp. 65–71, 1997.

[9] Sundeep Bajikar. Trusted Platform Module(TPM) based Security on Notebook PCs-White Paper. 2002.

[10] Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A general and flexible access-control system for the web. In *Proceedings of the 11th USENIX Security Symposium*, pp. 93–108, 2002.

[11] David W. Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with x.509 attribute certificates. *IEEE Internet Computing*, Vol. 7, No. 2, pp. 62–69, March 2003.

[12] Citrix Inc. Xen desktop. http://www.citrix.com/virtualization/desktop/xendesktop.html.

[13] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, pp. 46–66, 2001.

[14] Bram Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, pp. 68–72, 2003.

[15] Microsoft Corp. Limit the content that children can view on the web. http://windows.microsoft.com/en-US/windows-vista/Limit-the-content-that-children-can-view-on-the-web.

[16] Danga Interactive. memcached - a distributed memory object caching system. http://memcached.org/.

[17] Martin Dougiamas and Peter C. Taylor. Moodle: Using learning communities to create an open source course management system. In *Proceedings of the EDMEDIA 2003 Conference*, 2003.

[18] Ed. E. Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849, 2010.

[19] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur.*, Vol. 2, No. 1, pp. 34–64, February 1999.

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[21] Roy T. Fielding and Gail Kaiser. The apache http server project. *IEEE Internet Computing*, Vol. 1, pp. 88–90, 1997.

[22] Mozilla Foundation. Block and unblock websites with parental controls. http://support.mozilla.org/en-US/kb/block-and-unblock-websites-with-parental-controls.

[23] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, 1999.

[24] David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Commun. ACM*, Vol. 35, No. 2, pp. 97–107, February 1992.

[25] Eric Glass. The NTLM Authentication Protocol and Security Support Provider. 2003. http://davenport.sourceforge.net/ntlm.html.

[26] R. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, Vol. 7, pp. 34–45, June 1974.

[27] William T. Harding, Anita J. Reed, and Robed L. Gray. Cookies and web bugs: What they are and how they work together. *Information Systems Management*, Vol. 18, No. 3, p. 17, 2001.

[28] Hewlett-Packard Corp. et al. Advanced configuration and power interface specification. 2006.

[29] IEEE Standards Association. IEEE 802 Part3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. 1983.

[30] Intel Corporation. Preboot Execution Environment(PXE) Specification Version 2.1. 1999.

[31] Intel Corporation. Intel Trusted Execution Technology Preliminary Architecture Specification. 2007.

[32] Internet2 Middleware Architecture Committee for Education(MACE) Directory Working Group. Mace-dir. *http://middleware.internet2.edu/dir/*.

[33] Naomaru Itoi, William A. Arbaugh, Samuela J. Pollack, and Daniel M. Reeves. Personal secure booting. In *ACISP*, pp. 130–144, 2001.

[34] Brieuc Jeunhomme. Network boot and exotic root howto. http://tldp.org/HOWTO/Network-boot-HOWTO/.

[35] Jun Kanai, Mitaro Namiki, Kuniyasu Suzaki, and Toshiki Yagi. Mobile Thin-Client System with Fault Tolerance and Scalability by HTTP-FUSE-KNOPPIX-BOX. *International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA 2007)*, pp. 207–213, 2007.

[36] Takaaki Komura, Hiroaki Sano, Noritoshi Demizu, and Ken Makimura. Design and Implementation of Web Forward Proxy with Shibboleth Authentication. In *IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT)*, pp. 321–326, july 2011.

[37] Olga Kornievskaia, Peter Honeyman, Bill Doster, and Kevin Coffman. Kerberized credential translation: a solution to web access control. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, 2001.

[38] G. Kreitz and F. Niemela. Spotify – large scale, low latency, p2p music-on-demand streaming. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–10, 2010.

[39] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2109, 1997.

[40] Jian Liang, Rakesh Kumar, and Keith W. Ross. The kazaa overlay: A measurement study. *Computer Networks Journal (Elsevier)*, 2005.

[41] David Lie, Chandramohan A. Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 178–192. ACM, 2003.

[42] J. Linn. Generic Security Service Application Program Interface. RFC 2743, 2000.

[43] Kazutaka Morita. Sheepdog: Distributed storage system for qemu/kvm. In *LCA 2010 DS&R miniconf*, 2010.

[44] 中村めぐみ, 宗藤誠治, 須崎有康, 飯島賢吾, 八木豊志樹, 大澤一郎. トラステッド・コンピューティングによる HTTP-FUSE KNOPPIX クライアントのセキュリティ強化. 情報処理学会 研究報告, 第 2006-CSEC-34 巻, pp. 223–230, July 2006.

[45] T. Narten and R. Draves. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 3041, 2001.

[46] OASIS Standard. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. 2005. http://docs.oasisopen.org/security/saml/v2.0/saml-core-2.0-os.pdf Accessed: 2012/06/18.

[47] Chris M. O'Donnell. Using BitTorrent to distribute virtual machine images for classes. In *Proceedings of the 36th annual ACM SIGUCCS fall conference (SIGUCCS '08)*, pp. 287–290, 2008.

[48] Joon S. Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Trans. Inf. Syst. Secur.*, Vol. 4, No. 1, pp. 37–71, February 2001.

[49] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *First International Conference on Peer-to-Peer Computing*, 2001.

[50] Masatoshi Seki. dRuby and Rinda: Implementation and Application of Distributed Ruby and its Parallel Coordination Mechanism. *International Journal of Parallel Programming*, Vol. 37, No. 1, pp. 37–57, 2009.

[51] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 121–130, 2009.

[52] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *proceedings of the Winter 1988 Usenix Conference*, 1988.

[53] Richard Stern. Napster: A walking copyright infringement? *IEEE Micro*, Vol. 20, pp. 4–5, 2000.

[54] 須崎有康, 八木豊志樹, 飯島賢吾, 北川健司, 田代秀一. ネットワークに対応した分割圧縮ループバックデバイス HTTP-FUSE-CLOOP とそれから起動する Linux. インターネットコンファレンス 2005, Oct 2005.

[55] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, Hideyuki Tan, and Kazuhiro Goto. SFS-KNOPPIX. In *Fourth IEEE International Symposium on Network Computing and Applications (NCA 2005)*, pp. 247–250, 2005.

[56] Kuniyasu Suzaki, Toshiki Yagi, Kengo Iijima, and Nguyen Anh Quynh. OS Circular: InternetClient for Reference. In *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*, pp. 105–116, 2007.

[57] Shinichi Suzuki, Yasushi Shinjo, Toshio Hirotsu, Kazuhiko Kato, and Kozo Itano. Name-level approach for egress network access control. In *Networking - ICN*, Vol. 3421, pp. 284–296. 2005.

[58] Symantec Corporation. Norton ghost. http://us.norton.com/ghost.

[59] 高田真吾, 佐藤聡, 新城靖, 中井央, 板野肯三. 認証デバイスを用いた OS の起動・終了制御. 情報処理学会論文誌, Vol. 50, No. 3, pp. 1043–1053, March 2009.

[60] Shingo Takada, Akira Sato, Yasushi Shinjo, Hisashi Nakai, Koichi Sakurai, and Kozo Itano. A Simple Collaborative Method in Web Proxy Access Control for Supporting Complex Authentication Mechanisms. In *Proceedings of the 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2012.

[61] Shingo Takada, Akira Sato, Yasushi Shinjo, Hisashi Nakai, Akiyoshi Sugiki, and Kozo Itano. A P2P Approach to Scalable Network Booting. In *Proceedings of the Third International Conference on Networking and Computing*, 2012.

[62] Kenzi Watanabe, Makoto Otani, Shinichi Tadaki, and Yoshiaki Watanabe. Opengate on cloud. *International Conference on Advanced Information Networking and Applications Workshops*, pp. 1027–1030, 2012.

[63] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06)*, pp. 307–320, 2006.

[64] Wensheng Xu, David W. Chadwick, and Sassa Otenko. Development of a Flexible PERMIS Authorisation Module for Shibboleth and Apache Server. In *EuroPKI*, Vol. 3545 of *Lecture Notes in Computer Science*, pp. 162–179. Springer, 2005.

[65] L. Zhu, P. Leach, K. Jaganathan, and W. Ingersoll. The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism. RFC 4178, 2005.

[66] 金井遵, 須崎有康, 八木豊志樹, 並木美太郎. HTTP-FUSE-cloop による P2P モバイルシンクライアントシステム. 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol. 2007, No. 36, pp. 155–162, 2007.

[67] 古橋貞之. 分散システムのためのメッセージ表現手法に関する研究. 筑波大学大学院博士課程 システム情報工学研究科修士論文, 2012.

[68] 表祐志, 品川高廣, 加藤和彦. 仮想マシンモニタによる透過的ネットワークブート方式. 情報処理学会論文誌 コンピューティングシステム（ACS）, Vol. 4, No. 4, pp. 228–245, 2011.

著者文献リスト

この論文で述べられている研究内容は、以下の論文で発表済みである。

査読付き論文誌論文

1. 高田 真吾, 佐藤 聡, 新城 靖, 中井 央, 板野 肯三. “認証デバイスを用いた OS の起動・終了制御.” 情報処理学会論文誌, Vol.50, No.3, pp.1043-1052 (2009)

査読付き国際会議論文

1. Shingo Takada, Akira Sato, Yasushi Shinjo, Hisashi Nakai, Koichi Sakurai and Kozo Itano. “A Simple Collaborative Method in Web Proxy Access Control for Supporting Complex Authentication Mechanisms.” The 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing, 10 pages (2012)

2. Shingo Takada, Akira Sato, Yasushi Shinjo, Hisashi Nakai, Akiyoshi Sugiki and Kozo Itano. “A P2P Approach to Scalable Network Booting.” The Third International Conference on Networking and Computing, 7 pages (2012)

査読なし論文

1. 高田真吾, 佐藤聡, 新城靖, 中井央, 板野肯三. “認証デバイスを用いた OS の安全な起動制御.” 情報処理学会研究報告 2008-IOT-1, pp. 77-82 (2008)

2. 高田真吾, 佐藤聡, 新城靖, 中井央, 板野肯三. “認証デバイスを用いた OS の起動・終了制御システムにおける起動時間の短縮.” 情報処理学会研究報告 2009-IOT-4, pp. 155-160 (2009)

3. 高田真吾, 金子直矢, 齊藤剛, 佐藤聡, 新城靖, 中井央, 板野肯三. “UPKI 認証連携基盤を用いた Web アクセス制御.” 情報処理学会研究報告 2010-IOT-8, No.38, pp. 1-6 (2010)

4. 高田真吾, 佐藤聡, 中井央, 杉木章義, 新城靖. “レイテンシの違いを利用したネットブート配信環境の局所的な切り替え手法.” 情報処理学会研究報告 2011-IOT-16, pp. 1-6 (2012)