Windows PCによるOffice Grid

田中堅一 上原稔 村上真 山際基

東洋大学大学院 工学研究科 情報システム専攻

近年においてはハードウェア技術の進歩により、一般向けPCの性能も大きく向上している。しかしそれらハードウェア資源の多くは遊休資源となり、活用されていない。特にWindows PCについては、その世界的なシェアを考えれば膨大な資源が眠っている。その資源を活用するために我々はWindowsベースのグリッドについて研究を行っている。同時にビジネス分野においてもグリッド技術を利用することでコスト削減を行おうとする場合もあることから、本論文ではグリッドの応用としてオフィスグリッドについて述べる。研究はMicrosoft Officeを用いた文書フォーマット変換をグリッドによって高効率に行う技法について行った。オフィスソフトウェアをグリッド上から利用する方法、並びにグリッドを利用することの利点と課題について考察する。

Office Grid based on Windows PC

Kenichi Tanaka Minoru Uehara Makoto Murakami Motoi Yamagiwa

Toyo University Graduate School, Department of Open Information Systems

Recently, HW technology is evolved and the performance of PC is also increased. However, almost HW resources are idle. So, grid which uses such idle resources efficiently is proposed. However, Windows PC is not used for grid. So, we are studying an Windows based Grid (Windows Grid) in order to solve this issue. Windows Grid has no killer application. In this paper, we propose Office Grid which uses office application in Grid. We have developed a document converter of Microsoft office to plain text as an example. This application is usually used in search engine and so on. We describe how to implement and use Office Grid and finally evaluate it.

1. はじめに

ハードウェア技術の進歩により近年では、エンドユーザー向けPCにおいても高性能なCPUや大容量のディスクが搭載されるようになった。しかしながら、それら高性能なハードウェアの性能は完全に利用されているわけではなく、多量の遊休資源となっている。これら遊休資源を無駄にすることなく利用する手法の一つがグリッドコンピューティングである。特にビジネスや科学技術計算分野においてはコンピューティング需要が非常に高まっており、グリッドコンピューティングによって計算資源を確保することが必要となっている。

グリッドコンピューティングに関する研究はすでに多数行われ、その成果といえるものがグリッドミドルウェアである。グリッドミドルウェアにより今日では容易にグリッドコンピューティング環境を構築することが可能となっている。しかし、それらグリッドミドルウェアの成り立ちから、主にUNIXやLinux系統のOSでサポートされていることが多い。だが全世界におけるOSのシェアは、Windowsが90%を占めている。現状のグリッドミドルウェアではこれらの資源を利用することができないのである。したがって、Windowsベースの資源を利用できる、もしくはWindowsを中心として利用できるグリッド、Windows Gridの開発が必要となる。そのために我々はオープンソースのグリッドミドルウェアを用いて、Windows Gridの開発を行ってきた。結果は制限つきではあるがWindowsにおいてもグリッドは利用可能である。またマルチコアCPUの各コアを1つのCPUとみなすことで、仮想的に複数のPCであるかのように利用できることも判明した[5][6]。

またビジネスの分野ではオフィス文書の取り扱いに関して、グリッド技術を応用できる場合がある。特に文書ファイルの場合、外部へ提出する場合などでフォーマットの変換必要となる場面がある。各ファイルをひとつずつ変換する作業は非常に無駄な作業である。この作業をグリッド技術によって行えるようにすれば、短時間で多量の文書を変換できる。さらにMicrosoft Sharepointのような文書共有システムと連携することで、部署や個人の持つファイルの不整合をなくすことができる。総じて作業効率の向上が見込めるのである。

ところで、本学ではインターネットを利用したレポート提出システムを利用している[1]。このシステムによって科目や課題別にレポートの提出状況を管理することができる。このシステムではコピーされたレポートが提出されたとしても検知できない。そこで類似レポートの検出を行う研究を我々は行った[2][3]。文献[2]では、専用のレポート提出システムに類似判定機能を組み込んだ。文献[3]では、Moodle[4]に類似判定機能を組み込んだ。

レポートの類似性は、レポートの内の用語分布の類似度で判定する。用語分布の類似度は、各レポートの特徴語ベクトルの内積に等しい。この計算量は、レポート数をNとすると$O(N^2)$になる。また、特徴語ベクトルを求めるには、レポートファイル（主にMS-Word形式）をプレーンテキスト形式に変換し、形態素解析を行う必要がある。これらの処理の計算量は、定数的であるが、極めて大きく無視できない。

そこで、我々はWindows Gridを用いてOffice文書を処理するOffice Gridを開発した。このOffice Gridでは、Officeアプリケーションを Gridで制御することで、(Wordを含む) 多量のOffice文書を処理することができる。本論文では、特にWord文書のテキスト変換を主要アプリケーションとして評価を行う。

本論文の構成は以下の通りである。2章で関連研究について述べる。3章でOffice Gridについて述べる。4章では、Office Gridを評価する。最後に結論を述べる。

2. 関連研究

2.1 グリッドコンピューティング

グリッドコンピューティングは、ネットワークによって異なる地域や組織に属しているPCを相互接続し、仮想的な巨大コンピュータとして扱うための技術である。同様の技術にクラスタがあるが、グリッドでは一般的なPCを使用することが特徴である。そのため相互に関係しているタスクを分散処理させることには向いていないが、遊休状態にあるPCを利用することで計算資源を無駄にすることなく利用できる。

グリッドを構築する際にはミドルウェアを利用する。現在ではSun Grid EngineやApple Xgridなどが存在しているが、本研究ではGlobus Toolkitを使用した。Globus Toolkitはオープンソースプロジェクトとして開発されているミドルウェアであり、実質的な業界標準となっている。特徴はWebサービスベースでグリッドを構築する点である。グリッドを「状態をもつWebサービス」として公開することで、Webサービス分野の技術を利用することができる。

2.2 オフィスソフトウェア

オフィススイートとも呼ばれるこのようなソフトウェアは、業務で利用されるようなワープロ、表計算、プレゼンテーションなどのアプリケーションをまとめたパッケージソフトウェアである。現在Windows用ではMicrosoft Office (以下MS-Office)がほぼ市場を独占している状態である。しかしOpenOfficeやKOffice、GNOME Officeなど新しいオフィスソフトウェアも登場してきており、徐々にMS-Officeのシェアを奪いつつある。

本研究では利用者の人数が最も多いと考えられるMS-Officeを対象とした。

2.3 過去の研究

現状においてグリッドは基本的にLinuxベースの技術である。そこで世界シェア90%を持つWindowsベースPCをグリッドで利用できないかと考え、研究を行ってきた。しかしながらWindows PCであっても機能は制限されるものの、グリッドの利用は十分可能である[5][6]。またWindowsの資源を利用しつつ、完全なグリッドを使用したのであれば、仮想マシンを用いるという手法もある[7][8]。仮想マシンを用いた場合でも、性能面では劣ることはない。

これまでの研究でWindows PC資源をグリッドで利用することは可能となった。しかし、Windows固有の機能を使ったグリッドではなかった。本研究ではWindows独自の機能であるObject Linking and Embedding (OLE) を使用したOffice Gridの開発を行った。

3. Office Grid

グリッドは主に科学技術計算など、多量の計算を要求する場面で利用されている。このような場面以外に、グリッドの技術を利用するものとしてビジネス分野がある。ビジネス分野におけるグリッド (ビジネスグリッド)は、グリッド技術によってシステムの信頼性や運用性を向上するものである[11]。本研究では文書ドキュメントの運用について着目し、MS-Office Wordを対象として文書フォーマット変換を行うOffice Gridを構築することを目的とした。

ここでグリッドサービスが文書ファイルを扱うために、ファイル自体の転送についていくつかの方法がある。本研究では、クライアントからバイナリデータとしてグリッドサービスに対して送信する方法と、ある特定の共有スペースに文書を保管しておき、サービスに対して文書ファイルまでのURLを指示する方法を採用した。MS-OfficeはWindowsのネットワークパスによるファイルの参照が可能であるため、後者の方法はアプリケーション自体の機能を利用する。また前者の方法では、バイナリデータを転送する必要がある。Globusによる通常の方法ではバイト配列としてデータを読み込んだのち、転送時にシリアライザによってBase64によるエンコードが行われる。これによってHTTPベースのSOAP通信においてバイナリデータを送受信することができる。しかし、この方法では実行時のオーバーヘッドが大きくなることが予想される。そこで本研究ではもうひとつ、事前にBase64でエンコードしておき文字列として送信する手法での評価も行った。

以下に本研究で作成した2種類のサービスについて解説する。本サービスでは、MS-Wordの文書ドキュメントのテキストファイルへの変換を対象とした。どちらのサービスも最終的にOLEを用いてMS-Wordを呼び出し、ファイルの再保存によって変換を行っている。

3.1 Word2Text Service

このサービスはMS-Word文書ドキュメントをテキストファイルに変換するために作成したサービスである。MS-Word専用であり、さらにテキストファイルの変換のみ行うことができる。そのかわりシステム自体は簡素な構造となっている。

このサービスではクライアントから送信された文書ドキュメントをOLEによってMS-Wordで一度開かせる。その後MS-Wordによって一時ディレクトリにテキストファイルとして保存を行う。最後に保存されたテキストファイルを文字列として、クライアントに送信する。OLE呼び出しはJCom

(Java-COM Bridge, [10]) ライブラリを使用している。なおサービスの仕様上文書ドキュメントは ZIP で書庫化された状態で送信される必要がある。これは複数の文書ドキュメントを扱うための仕様である。そのためクライアント側では、文書ドキュメントを ZIP ファイルとしてまとめる処理を行う。図 1 は Word2Text Service のシステム概要を図示したものである。

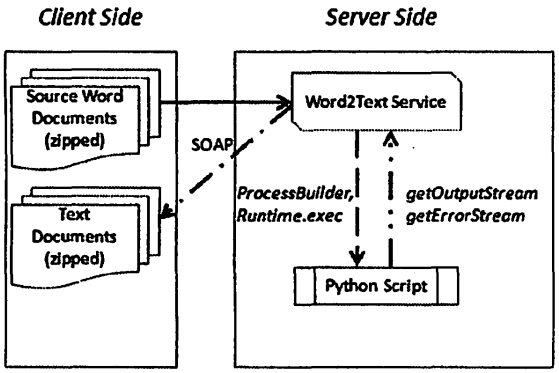

図 1 Word2Text Service

3.2 Task Management Services

Task Management Services は、Word2Text Service とは異なり、タスクベースの汎用的なジョブ実行サービス群である。現在は Task Management Service と Task Execution Service の 2 つから構成されているが、現段階で実装済みは Task Execution Service のみである。図 2 に Task Management Services の関係図を示す。

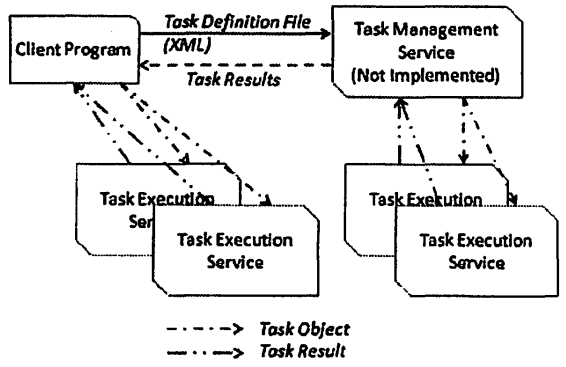

図 2 Task Management Services

このサービス群ではタスク単位で各ノードにジョブを割り当てる。タスクにはジョブを複数設定でき、これらのジョブは先に定義されている順に実行される。これは一種の依存関係を表現するものであり、順番に処理されなければならないジョブを割り当てるための仕様である。

各ジョブは、自身を識別する ID と処理を行うプログラム名、およびプログラムへの引数のリストを持つ。引数は文字列のみを許可しているため、ファイルを渡す場合は Base64 によるエンコードをする必要がある。またファイル名とそのデータを保持するオブジェクトをタスクに割り当て、そのファイル名を引数とする方法も用意してある。この方法を応用すると、プログラムにスクリプト実行系を指定し、引数に実行するスクリプトを指定することで任意のプログラムを実行させることが可能である。本研究ではこの機能を利用し、Python スクリプトと pywin32 拡張を使用して文書ドキュメント変換を行っている。次に Task Execution Service で行われるタスクスケジューリングの流れを示す。

1) 作業用ディレクトリを作成する。
2) タスクに添付されたファイルの復元を行う。
3) 標準出力、標準エラー出力記録用の一時ファイルを作成する。
4) コマンドラインの構築を行う。
5) 実行日時を定義する Calendar オブジェクトをミリ秒へと変換する。
6) 実行日時と現在時刻の差を取り、現在からの遅延時間を計算する。このとき実行日時が現在よりも過去なら遅延は 0 ミリ秒とする。
7) タスク実行スレッドを作成する。
8) ScheduledThreadPoolExecutor にタスク実行スレッドと遅延時間を指定してスケジューリングする。指定された遅延時間後にスレッドが有効化される。

タスク定義は、クライアント-サービス間では Java オブジェクトとして扱われる。したがってクライアント側がユーザ定義のタスクを Java オブジェクトに変換する必要がある。今回はタスク-ジョブの関係や各ジョブに割り当てられるプログラムの関係を表現するために、XML によってタスク定義を行った。図 3 にタスク定義の一部を示す。

```
<xml version="1.0" encoding="UTF-8">
<tasklist>
  <task name="Hello"
        date="2008/05/04 12:00"
        delay="0">
    <job name="Echo Hello" id="0">
      <program command="cmd">
        <argument type="0">/C</argument>
        <argument type="0">
          &amp;quot;echo Hello&amp;quot;
        </argument>
      </program></job></task>
                    ...
```

図 3 タスク定義

4. 評価

本研究で行った評価の種類と、使用したサービスの関係を表 1 に示す。なお、ファイル送信はクライアントから直接サービスに対してファイルデータを送信する方法である。このとき Base64 エンコードをどの時点で行うかによって二種類の評価を行った。ネットワークパスは、Windows で利用可能なパス指定方式である。その形式は

¥¥*<host name>*¥*<path>*

である。*<host name>*にはホスト名もしくはホストの IP アドレスを指定し、*<path>*には共有ディレクトリからのパスを指定する。MS-Word はこのネットワークパスによるファイル参照が可能であるので、その機能を利用した。

評価自体は、ネットワーク遅延を無視するためにすべてローカルマシン内で行った。そのため得られた結果はほぼ純粋にサービスの性能と言える。表 2 に評価に使用した環境を示す。また表 3 に、すべての評価で使用した文書ドキュメントのファイルサイズと文字数を示す。

表 1 評価とサービス

評価	サービス
ファイル送信 (実行時エンコード)	Word2Text
ファイル送信 (事前エンコード)	Task Management
ネットワークパス	Task Management

表 2 評価環境

OS	Windows XP Professional x64 SP2
CPU	AMD Athlon 64 X2 Dual Core Processor 3800+, 2.00 GHz
Memory	3.93 GB RAN
MS-Office	2007 (PDF / XPS Addin)
Language Runtime	Java / 1.6.0 Update 5 Python / 2.5.1

表 3 評価に使用した文書ドキュメント

Doc.	File Size (Bytes)	#chars
DOC 01	162,304	3,288
DOC 02	258,048	2,887
DOC 03	486,912	10,765
DOC 04	902,144	25,044
DOC 05	2,128,384	8,096

4.1 実行時エンコード方式

実行時エンコード方式はバイナリデータを XML データへと変換する作業を、サービスに要求を出す段階で行う方式である。実際の動作はクライアントがサービスへのインターフェイス (Port Type と呼ばれる) に対して要求を出す際、すべてのオブジェクトを XML シリアライズによって XML データに変換することで、バイナリデータを扱うことができるようにしている。

表 4 は実行時エンコード方式による変換処理の評価結果である。結果はすべてミリ秒単位であり、クライアント側で取得した経過時間である。また DOC 05 に関してはエラーが発生し、有効な値が取得できなかった。そのため結果は省略している。表中の ZIP、Submit、Convert、Misc はそれぞれドキュメントの圧縮、圧縮されたドキュメントの送信、テキストへの変換処理、その他の処理時間である。

表 4 からまず、ファイルサイズに比例して総合的な処理時間が増加していることがわかる。しかし実際の変換処理はファイルサイズに比例しているわけではなく、さらに文字数に対しても比例はしていない。この傾向は数回繰り返しても変化はなかった。

注目するべきは送信処理である。ここはファイルサイズに比例する形で大幅に増加傾向である。この時間が総合的な処理時間に影響を及ぼしていると考えられる。

この評価では、このサービスの重大な問題点が発覚した。その問題は、SOAP メッセージを解析する XML パーサが、多量のメモリを消費するという問題である。この XML パーサは Globus によって提供されているため、他のパーさへの切り替えは難しい。実際の評価時には DOC 04 の評価段階で、Java VM のヒープ割り当てを 256 MB まで増やして状態で評価が可能であった。さらに DOC 05 の評価時には、1024 MB (1 GB) のヒープを割り当てても OutOfMemoryError によって評価が行えなかった。ゆえに、Globus の機能を利用している実行時エンコード方式は実用的ではないという結果が得られた。そこで次に開発した Task Management Service では実行時にバイナリデータの変換を行わないことで、メモリ使用量の削減を試みた。

表 4 評価結果 1

Doc #	01	02	03	04
ZIP [ms]	63	31	109	219
Submit [ms]	7815	31234	61641	406859
Convert [ms]	21265	10625	578	672
Misc [ms]	6235	5829	6110	6141
Total [ms]	35438	47719	68438	413891

4.2 事前エンコード方式

4.1 の評価では、Globus の提供する XML パーサが多量のメモリを消費することで OutOfMemoryError を発生させることとなった。しかし、先の結果からは XML パーサが消費しているかどうかまではわからない。つまり、変換されたバイナリデータのサイズによって発生している可能性もある。そこで、バイナリデータ自体を事前に Base64 によって文字列へと変換しておく方式での評価を行った。この方式では Task Management Service を用いる。この方式による評価結果を表 5 に示す。結果はすべてミリ秒で計測した。なお評価時に Python によるプログラムもファイルデータとして転送している。これはサービスの設計による制限である。また表中の Schedule は表 4 における ZIP と Submit に相当する処理である。

この評価では OutOfMemoriError は発生しなかった。したがって 4.1 の評価時のエラーは XML パーサによる Java オブジェクト処理が引き起こしていたと考えられる。表 5 の結果の中で、DOC 01 の評価を除けば、タスクのスケジュールと文書変換処理の

どちらもファイルサイズ比例する形となっている。

表5 評価結果 2

	Schedule	Convert	Total
DOC 01	375	9,859	10,906
DOC 02	500	2,046	3,297
DOC 03	515	2,250	4,093
DOC 04	766	2,391	5,047
DOC 05	1,563	2,641	5,266

4.3 ネットワークパス

この評価は4.2の評価に加えて行ったものである。4.2ではファイル自体を転送していたが、本評価では共有ディレクトリに配置し、ネットワークパスを通じて参照する方法を用いる。各ジョブの引数にはネットワークパスを与え、ファイルデータはタスクには含まれないようにしている。評価に使用したプログラムは4.2と同一である。評価結果を表6に示す。単位はミリ秒である。

全体的な傾向は 4.2 の結果と同様である。若干変換処理時間が延びているが、これはネットワークパスを使うことによるオーバーヘッドと考えられる。

表6 評価結果 3

	Schedule	Convert	Total
DOC 01	281	11,860	12,750
DOC 02	250	2,641	3,360
DOC 03	266	2,797	4,219
DOC 04	235	2,813	4,688
DOC 05	297	3,141	4,453

5. 考察

5.1 Word2Text Service についての考察

Word2Text service では、実行時に Globus の機能を利用してバイナリデータを XML 変換する手法をとった。この方式ではクライアントからサービスへ文書ドキュメントを送信する処理が、最も大きなオーバーヘッドなっている。クライアント側におけるバイナリデータの XML への変換処理は、サービス構築の際に自動生成されるスタブによって実行される。このスタブ内で多量のオブジェクトが作成されメモリ領域を消費すると考えられる。またサービス側での復号時には、オブジェクトの量に比例して処理時間がかかることは明らかである。

5.2 Task Management Services についての考察

このサービスでは、事前エンコード方式とネットワークパス利用方式の二通りの評価を行った。結果は若干ネットワークパス利用の方がタスク完了までの時間が長いが、Word2Text Service を利用した場合よりもはるかに高速である。図 4 は各手法の総合的な処理時間の比較を示したものである。なお実行時変換、事前エンコード、ネットワークパスをそれぞれ Runtime、Encoded、Network と図中では表記している。また DOC 05 によるデータは実行時変換では計測できなかったため省略している。

実行時変換処理の場合、ドキュメント数の増加に伴いタスクの送信処理時間が大きく増加したために、全体処理時間も増加している（表 4）。これはバイナリファイルをバイト列として読み込んだ後、送信処理中で base64 のエンコード処理が行われていることが原因と考えられる。事実、事前にエンコード処理を行っている場合はタスク送信にかかる時間はわずかである（表5）。

ところで DOC 01 に関しては、MS-Word がドキュメント開く際の処理が他のドキュメントと比較して大きな負荷となっていると考えられる。特に元のサイズが大きい画像を持っていると遅くなるようである。また 4.1 と比較して Convert 処理が大きく短縮されている原因として考えられるのは、プログラミング言語とライブラリである。4.1 では Java による OLE ブリッジを使用したが、4.2 では Python による Win32 呼び出しを使用している。そのライブラリの実装に大きく影響を受けたと推測されるが、この点についは未確認である。

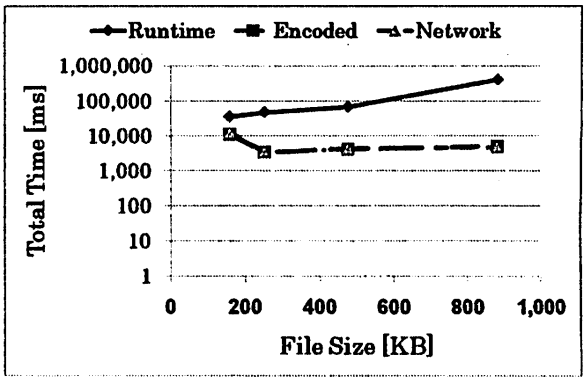

図 4 手法ごとの総合処理時間比較

5.3 実存データによる並列変換

これまでの評価は単一のファイルを対象としたものであった。本評価は本学のレポート提出システムで提出されたデータを使用し、複数のサービスによる並列変換を対象とするものである。表 7 は実験に用いた文書ドキュメントのファイルサイズである。これらの文書ドキュメントについてコンテナ数を 2、4、8 台で並列に変換を行った。そして変換時の 1 ファイルあたりの平均処理時間を図 5 に示す。なお、評価には事前エンコード方式を用いた。加えて、評価に使用した PC の性能は一律ではなく、表 8 のようになっている。各 PC 上で 2 つのサービスコンテナを動作させることで、仮想的に 2 台分として扱っている。本評価では 4 章 3 節で使用した Python スクリプトではなく、同じ動作をする Ruby スクリプトを使用した。

Task Execution Service 内では ScheduledThreadPoolExecutor によってタスクのスケジューリングを行っている。スレッドプールのサイズは 10 固定である。したがって各コンテナでは

同時に 10 このスレッドが動作する可能性がある。本評価時には最大 8 コンテナを動作させているため、理論上最大 80 スレッドが同時に動作する。しかし PC のリソースには限界があり、また Java 仮想マシン内で 80 ものスレッドが同時に動作することとなれば、相当な負荷となる。128 の文書ドキュメントを変換時の平均速度が低下しているのは、スレッド数が多すぎるためにスレッド切り替えのオーバーヘッドが大きく影響したためと考えられる。またドキュメント数 8 の場合の速度が非常に遅くなっているが、他のデータを見る限りサービスの性能ではなく MS-Word に起因するものによると考えられるが、現状では原因不明である。

ドキュメント数 16 以上の平均処理時間はコンテナ数 2 の時におよそ 3 秒、コンテナ数 8 の場合はおよそ 1 秒となっている。最も早い時で 0.7 秒であった。OLE 呼び出しのオーバーヘッド、ならびにスクリプトの実行などを考慮すると、これ以上の処理時間短縮は難しい。なお全体の処理時間はドキュメント数 8 の場合を除いてドキュメント数に比例し、コンテナ数に反比例する傾向となっている。とはいえ、ドキュメント数が多いほど処理時間がかかるのは当然といえる。本評価ではコンテナ数 8 の時に 128 ドキュメントの変換に 172 秒かかっている。

表 7 評価結果 4

#docs	Min [B]	Max [B]	Average [B]
8 docs	104,448	385,536	219,520
16 docs	46,080	982,016	259,168
32 docs	70,656	273,920	130,784
64 docs	55,296	292,352	122,768
128 docs	46,592	649,216	165628

表 8 評価マシン

	PC 01 / 02	PC 03	PC 04
OS	Win XP Pro x64 SP2	Win XP Pro SP2	Win XP Home SP2
CPU	Athlon X2 3800+, 2.00 GHz	Athlon X2 4600+, 2.41 GHz	Athlon X2 5200+, 2.61 GHz
MEM	3.93 GB	1.93 GB	3.25 GB

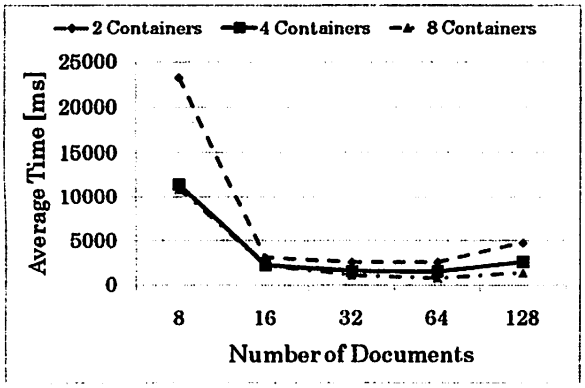

図 5 ドキュメントあたりの平均処理時間

6. まとめ

本研究では Windows Grid の応用としてオフィス文書のフォーマット変換を行うオフィスグリッドの構築を行った。評価では文書ドキュメントをテキスト文書へと変換を行った。テキストへの変換自体はすでに Antiword[9]と呼ばれるソフトウェアによって可能である。事実テキストへの変換処理速度では本サービスは Antiword と比較して非常に遅いといえる。しかしオフィスグリッドとしての利点は、テキスト以外のフォーマットへの対応である。特に Task Execution Service で使用したスクリプトは、OLE によって MS-Word を呼び出す形式上、MS-Word の対応するフォーマットであれば自由に変換が可能である。また MS-Office が新しくなったとしても OLE の使用が変わらない限り、プログラムを大幅に変える必要がなく更新が簡易に行える。

課題は、多様な形式への対応である。現状では MS-Word でのみ評価を行ったが Excel や Powerpoint、さらには OpenDocument などに対応することが必要である。特に OpenDocument へ対応すれば OpenOffice など Windows 以外の環境との文書の交換が可能となり、非常に使いやすいものとなる。

参考文献

[1] 萩原 翔太, 上原 稔, "オープンソースを用いた Web 情報システムの開発 －多様な機能を持ったレポートシステム－", 東洋大学第 7 回工業技術研究所講演会, pp.65-66, (2006.2.23)

[2] 上原稔 "レポートの類似度判定に関する研究", 東洋大学第 9 回工業技術研究所講演会, pp.66-67, (2007.2.22)

[3] 沼田 千秋、上原 稔 "類似レポート検出システム", 東洋大学第 11 回工業技術研究所講演会, pp.77-78, (2008.2.21)

[4] Moodle – A free, Open Source Course Management System for Online Learning, http://moodle.org/

[5] 田中堅一、上原稔、森秀樹、"オープンソース Windows グリッドによる CG の並列計算"、マルチメディア通信と分散処理ワークショップ, pp.55-60

[6] Kenichi Tanaka, Minoru Uehara, Hideki Mori, "Parallel Computing of CG using Open Source Windows Grid", FINA 2008

[7] 田中堅一、上原 稔、森 秀樹、「Windows Grid と VM における Linux Grid についての考察」、情報処理学会 第 70 回全国大会

[8] Kenichi Tanaka, Minoru Uehara, Hideki Mori, "A Case Study of a Linux Grid on Windows using Virtual Machine", FINA 2008

[9] Antiword: a Free MS Word document reader, http://www.winfield.demon.nl/

[10] SourceForge.net: JCom (Java-COM Bridge), http://sourceforge.net/projects/jcom/

[11] 「特集 ビジネスグリッドコンピューティング」、情報処理 第 47 巻 第 9 号 pp.945-985