(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5349481号
(P5349481)
(45)発行日　平成25年11月20日(2013.11.20)　(24)登録日　平成25年8月30日(2013.8.30)

(51)Int.Cl.　F I
G06F 3/12 (2006.01)　G06F 3/12 Z
G06F 3/12 C

請求項の数 3　(全 22 頁)

(21)出願番号　特願2010-525808 (P2010-525808)
(86)(22)出願日　平成20年9月12日 (2008.9.12)
(65)公表番号　特表2010-541041 (P2010-541041A)
(43)公表日　平成22年12月24日 (2010.12.24)
(86)国際出願番号　PCT/US2008/010658
(87)国際公開番号　WO2009/038670
(87)国際公開日　平成21年3月26日 (2009.3.26)
審査請求日　平成23年9月12日 (2011.9.12)
(31)優先権主張番号　11/858,477
(32)優先日　平成19年9月20日 (2007.9.20)
(33)優先権主張国　米国 (US)

(73)特許権者　590000846
イーストマン　コダック　カンパニー
アメリカ合衆国　ニューヨーク州　ロチェスター　ステート　ストリート　３４３
(74)代理人　100075258
弁理士　吉田　研二
(74)代理人　100096976
弁理士　石田　純
(72)発明者　アロンシュタム　ボリス
アメリカ合衆国　ニューヨーク　ロチェスター　ステート　ストリート　３４３
(72)発明者　カイン　レオニド
アメリカ合衆国　ニューヨーク　ロチェスター　ステート　ストリート　３４３

最終頁に続く

(54)【発明の名称】ページ記述言語の並行処理

(57)【特許請求の範囲】
【請求項１】
ページ独立性を欠くページ記述言語（ＰＤＬ）で記述された印刷ジョブを編成する方法であって、前記編成されたジョブはページ独立である必要がなく、複数のプロセッサにより効率的に分割および処理可能であって、
ＰＤＬジョブに対して構文解析パスを１回実行するステップと、
ＰＤＬジョブプロデューサを検出するステップと、
前記ＰＤＬジョブ内の共通リソースを検出して印付けするステップと、
前記ＰＤＬジョブ内のページ境界を検出して印付けするステップと、
オリジナルの前記ＰＤＬジョブについての前記検出を行うステップ群に従って、前記ＰＤＬジョブ内のデータおよびリソースを再配置することなく、前記共通リソースの印及び前記ページ境界の印を含む編成された表現を生成するステップとを含む方法。
【請求項２】
ページ独立性を欠くページ記述言語（ＰＤＬ）で記述された印刷ジョブの順序替えのための方法であって、
ＰＤＬジョブに対して構文解析パスを１回実行するステップと、
ＰＤＬジョブプロデューサを検出するステップと、
前記ＰＤＬジョブ内の共通リソースを検出して印付けするステップと、
前記ＰＤＬジョブ内のページ境界を検出して印付けするステップと、
各ページのグラフィック状態を規定するコマンドを記録するステップと、

オリジナルの前記ＰＤＬジョブについての前記検出を行うステップ群に従って、前記ＰＤＬジョブ内のデータおよびリソースを再配置することなく、前記共通リソースの印及び前記ページ境界の印を含む編成された表現を生成するステップと、

前記リソースを実行するステップと、

グラフィック状態コマンドを、順序替えして送出される前記ページ群の先頭に置くステップとを含む方法。

【請求項３】

並べ替えがページ反転である、請求項２に記載の方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、印刷システム、ディスプレイシステム、ＰＤＬ解析システム、およびＰＤＬ変換に必要とされるページ記述言語（ＰＤＬ）データの効率的な処理方法および装置に関する。

【背景技術】

【０００２】

ポストスクリプト（登録商標）言語は当業者によく知られている。ポストスクリプトは、印刷ジョブにおいてページの記述に用いるコマンドの豊富な組を含むページ記述言語（ＰＤＬ）である。ポストスクリプトと他のＰＤＬ、例えばＩＰＤＳ、ＰＤＦ、ＰＣＬ、ＰＰＭＬの主な違いは、ポストスクリプトがプログラミング言語である点である。これにより、ページコンテンツを表す際の表現力および柔軟性が向上するが、柔軟性の代償は高い。一般のポストスクリプトジョブでページを解釈するのは容易でない。ページを正しく解釈したりポストスクリプトジョブの有意味な変換を実行したりするためにポストスクリプトのインタプリタ（解釈機能）が必要である。Ａｄｏｂｅ社のコンフィギュラブル・ポストスクリプトインタプリタ（ＣＰＳＩ）はポストスクリプトインタプリタの一例であり、ポストスクリプトジョブを処理してビットマップを生成する。ＡｄｏｂｅＤｉｓｔｉｌｌｅｒはポストスクリプトインタプリタの別の例あり、ポストスクリプトジョブを処理してビットマップではなくＰＤＦファイルを生成する。

【０００３】

１９８４年にポストスクリプトが登場して以来、世界中の技術者は、ポストスクリプト言語における公知の限界を克服すべく多くの技術を実現してきた。これらの限界として以下のものがある。

ａ）プリンタに合わせた速度でのポストスクリプトジョブの実行を阻害する速度限界。

ｂ）複数の中央演算処理装置（ＣＰＵ）でページを並行処理するために必要とされるように、ポストスクリプトを別個の独立したページに分割できないこと。

ｃ）選択的なページ範囲の再印刷で必要とされるように、選択されたページを効率的に印刷できないこと。

以下に開示する本発明並びに、具体的な性能問題および実施おけるに共通的な傾向を理解するために、典型的なポストスクリプトインタプリタの説明が必要である。ポストスクリプトジョブの処理は、（多くの場合重なり合う）二つの段階、すなわち解釈段階および出力段階からなる。

－ポストスクリプトはインタプリタ言語である。任意の種類のインタプリタ（例：Ｐｅｒｌ、Ｊａｖａ（登録商標））と同様に、解釈を実行する間にポストスクリプトジョブが構文解析されて内部ジョブ構造が生成される。この内部ジョブ構造は、高レベルまたは低レベルのグラフィカルオブジェクトのリンクされたリスト（または木）、ジョブ内のページを記述する複合状態、または他の任意の独自の表現形式であってよい。

－出力段階において、内部ジョブ構造が処理されて必要な出力が作成される。印刷システムの場合、ページがレンダリングされてラスタ（例：生のビットマップ）が生成され、通常はプリンタに送られる。ＡｄｏｂｅＤｉｓｔｉｌｌｅｒの場合、ＰＤＦファイルが生

成される。他の形式（例：ＡＦＰ／ＩＰＤＳ）もまた同様の方法を用いて生成することができる。

【０００４】

従来、解釈は軽負荷の処理と考えられていた一方で、レンダリングは生成されるデータの量に応じて重負荷の処理と考えられていた。テキストおよびグラフィックを含むポストスクリプトページの典型的なソースデータは１００ＫＢ以下である。６００×６００ｄｐｉＣＭＹＫでレンダリングされた場合、典型的な未処理ビットマップページは１００ＭＢ以下、すなわちソースデータより１０００倍大きい。

【０００５】

上述の理由により、ポストスクリプト言語が登場して以来、技術者はレンダリングをスキップすべく「ヌル（Ｎｕｌｌ）デバイスへの書き込み」技術を用いてきた。この技術は、Ａｄｏｂｅ社「ポストスクリプト言語リファレンスマニュアル」("PostScript Language Reference Manual")の全バージョンに記述されている。この技術によれば、ヌルデバイスを設定し、次いで実デバイスを再立ち上げしてレンダリングを再開することによりページのレンダリングをスキップすることができる。ヌルデバイス方式は通常、解釈のオーバーヘッドを更に減らすために複数のポストスクリプトオペレータ（例：ｓｈｏｗ、ｉｍａｇｅ等）の再定義により強化される。このヌルデバイス方式を用いて、ページを解釈してレンダリングをスキップすることにより、ページをスキップすることができる。このようなスキップの仕組みを用いて、当業者は図１に示すようにページの並行処理を実施できる。

【０００６】

図１に４個のプロセッサを示す。この方式において、４個のプロセッサの各々が全体的なポストスクリプトジョブ１１を受け取り、各プロセッサは一部のページをスキップして他のページを処理する。例えば、第１のプロセッサ１２はページ１、５、９．．．を処理する一方、第２のプロセッサ１３はページ２、６、１０．．．を処理し、第３のプロセッサ１４はページ３、７、１１．．．．を処理し、第４のプロセッサ１５はページ４、８、．．．１２を処理する。明らかに、この単純な負荷バランシングアルゴリズムは、各プロセッサの現在の負荷、ページの複雑さ、および他の特徴を考慮に入れて改善できる。このような負荷バランシングの考察は後の図面全てに適用できる。

【０００７】

この方式から得られる利点は容易に分かる。単一ＣＰＵシステムがジョブ全体を処理するのに１００秒かかるものと仮定する。更に、解釈はレンダリングより４倍速いと仮定する（これはかなり合理的な仮定である）。これらの仮定によれば、解釈は２０秒で済む一方、レンダリングには８０秒かかる。図１を再び参照するに、各プロセッサは、同じ２０秒を解釈に費やす（各プロセッサがジョブ全体を解釈する必要がある）が、レンダリングには２０秒しか費やさない（各プロセッサはページの４分の１しかレンダリングする必要がない）。この場合、ジョブ全体は４０秒で処理される。これにより２．５倍の性能向上（１００／４０＝２．５）が達成される。

【０００８】

図２に８個のプロセッサを示す。処理は、別個のプロセッサを用いて解釈とレンダリングに分割される。インタプリタ２２は、解釈されたポストスクリプトストリームをレンダラ（レンダリング機能）２６へ受け渡すことにより、（上述のページパラレリズム（並列処理）に加え）パイプラインパラレリズムを実現する。上述の数を用いて、解釈段階とレンダリング段階がパイプライン化されている（並行して動作する）ことを考慮すれば、ジョブ全体は約２０秒で処理される。これにより５倍の性能向上（１００／２０＝５）が達成される。

【０００９】

図２に示す方法は、初期の印刷方式を用いていた時代のように解釈時間がレンダリング時間に比べて重要でなかった場合には全く充分であった。しかし１９８４年以降、解釈とレンダリングのバランスは以下の要因により大幅に変化している。

ａ）各社は高速化を目指して、極めて効率的なレンダリングシステムおよび独自のハードウェアソリューションを提供することにより、レンダリング技術に相当の投資を行なってきた。

ｂ）マルチＣＰＵシステムが極めて安価になった。主流のＣＰＵ技術において現行の汎用ＣＰＵは独立ＣＰＵとして動作する複数の処理コアを含んでいるのが最新の傾向である。近い将来、８コア、１６コア、および３２コアのＣＰＵの出現が期待される。

ｃ）今日、ますます多くのジョブが、重負荷の解釈処理を必要とする極めて複雑なグラフィックスおよび膨大な画像を含んでいる。

ｄ）印刷速度が大幅に向上しており、１００ｐｐｍ（ページ／分）を超え、更には１００ｐｐｍにも達している。

【００１０】

上述の要因の結果、各プロセッサがポストスクリプトジョブ全体を解釈するヌルデバイスへのレンダリングでは、高いエンジン速度を実現するには不十分となる。換言すれば、本質的に逐次的処理である解釈は印刷システムのボトルネックになる。例えば、各インタプリタはジョブの解釈に同じく２０秒を費やす必要があるため、図２にプロセッサを追加しても性能は向上しない。

【００１１】

図２における複数のインタプリタが各ジョブを複製することを理解すれば、当業者は図３に示すように解釈を切り離して別個のプロセッサに移すことができる。この図において、集中型解釈実行プロセッサ３２はポストスクリプトジョブ１１を解釈して、独立ページ３３を含むいくつかの内部ジョブ構造（ディスプレイリスト）を生成する。独立ページ３３のディスプレイリストは、個々のレンダリングプロセッサ３４へ送られる。図２に示す方式と比較して、本方式の主な利点は、同じ性能を実現するのに５個のＣＰＵしか必要としない点である。更に、集中型解釈実行プロセッサ３２としてより強力なＣＰＵを用いることにより、解釈のボトルネックをいくぶん緩和することができる。

【００１２】

この方式の重大な短所はその複雑さにある。すなわち、ポストスクリプトプロセッサを、別個のノードで動作している独立したインタプリタとレンダラとに分離するのは複雑な手順である。これには大幅なコード変更を要し、ソースコードがその変更を実行する必要がある。しかしこの方式の主な短所は、依然としてインタプリタがボトルネックであることである。上の例で提案された数にレンダリングプロセッサ３４の数を増やしても性能は向上しない。

【００１３】

これまで見てきたように、ボトルネックとしてのインタプリタを除去することによりシステムの全体的な速度を高める方法および装置を提供することが望ましい。更に、インタプリタの変更を必要としない方法および装置を提供することが望ましい。

【００１４】

集中型解釈方式の公知バリエーションとして図４に示すＰＤＦ方式がある。この方式では、ＰＳ／ＰＤＦコンバータ４２によりポストスクリプト（ＰＳ）ジョブ１１がＰＤＦに変換される。生成されたＰＤＦ４３はＰＤＦディストリビュータ（分配機能）４４により複数のプロセッサ４５に分配される。ポストスクリプトをＰＤＦに変換すべく利用可能な多くのユーティリティが存在する。ＡｄｏｂｅＤｉｓｔｉｌｌｅｒは恐らく最もよく知られているであろう。

【００１５】

また、ＰＤＦジョブ４３をプロセッサ４５に分配する多くの方式がある。

ａ）全てのプロセッサにＰＤＦファイル全体を受け渡すことができる。各プロセッサはどのページをレンダリングすべきかが指示される。

ｂ）ＰＤＦジョブを一連の単一ページＰＤＦファイルに変換することができる。これらの単一ページＰＤＦファイルをプロセッサへ受け渡すことができるため、各プロセッサはレンダリングが必要なＰＤＦページだけを受け取る。

ｃ）ＰＤＦを一連の単一ページポストスクリプトファイルに変換することができる。これらの単一ページポストスクリプトファイルをプロセッサへ受け渡すことができるため、各プロセッサはレンダリングが必要なポストスクリプトページだけを受け取る。

ｄ）単一のＰＤＦまたはポストスクリプトのページではなく、ページチャンク（複数ページからなる塊）を生成して必要なプロセッサに分配することができるため、単一ページの潜在的なリソースオーバーヘッドが減る。そのバリエーションとして、ジョブ全体をプロセッサの数に等しい個数の部分（本例では４個の部分）に分割することができる。

【００１６】

これらのＰＤＦ方式は、実行可能な方法であって当業界では公知である。同時に、これらには上で議論した「集中型解釈」方式と同じ主要な短所がある。すなわち、ＰＳ／ＰＤＦコンバータがポストスクリプトインタプリタであるため、当該コンバータがボトルネックとなる。更に、ＰＤＦへの変換によりかなりの余分なオーバーヘッドがコンバータに追加されるため、更により大きいボトルネックが生じることが知られている。このボトルネックのため、プロセッサを追加してもシステムの性能が向上しない。

【００１７】

再び図４を参照するに、ポストスクリプトからＰＤＦへの変換ではない、他の変換が可能である。例えば、ポストスクリプトからポストスクリプトへ、ポストスクリプトからＡＦＰへ、またはポストスクリプトからＸＰＳへの変換である。しかし、このようなコンバータは全てポストスクリプトインタプリタのインスタンスであるため、それら全てに上で議論した「集中型解釈」方式および「ＰＤＦ方式」と同じ重大な短所がある。すなわち、コンバータがボトルネックになるため、プロセッサを追加してもシステムの性能が向上しない。

【００１８】

これまで見てきたように、ボトルネックとなるインタプリタを除外することによりシステムの全体的な速度を上げる方法および装置を提供することが望ましい。更に、ポストスクリプトからＰＤＦその他の言語への変換を回避する方法および装置を提供することが望ましい。

【００１９】

ポストスクリプトジョブの構造化されていない性質に関する問題を認識した上で、Ａｄｏｂｅ社は既に１９８６年に「Ａｄｏｂｅ文書構造化規約仕様バージョン１」（ＤＳＣ仕様）を発行している。最も良く知られているＤＳＣ仕様バージョン３．０は１９９２年に発表された。この仕様には「並行印刷」と名付けられた独立したセクションがある。これは、ページの並行印刷がＤＳＣ準拠ポストスクリプトジョブの意図の一つであることを示している。

【００２０】

ＤＳＣ仕様は、ＰＳリソースの構文解析が簡単に行なえてページを再配置できるようにするタグの組を定義する。同仕様は更に、プロデューサ（生成機能）が「％！ＰＳ－Ａｄｏｂｅ３．０」を出力したならば、このポストスクリプトファイルがＤＳＣ準拠であることを保証することを義務付ける。残念ながら現実には、ほぼ全ての主要なポストスクリプトプロデューサが「％！ＰＳ－Ａｄｏｂｅ３．０」を挿入しているものの、これらのファイルがＤＳＣ準拠であることは稀である。

【００２１】

上記にもかかわらず、実行すれば分かるように、ＤＳＣコメントおよびプロデューサ固有のパターンを探して構文解析することにより、ポストスクリプトジョブの巨大な集合をうまく独立ページに分割することができる。この処理は複雑であるが、１９８８年以降複数の企業がこの方式をうまく活用している。例えば、Ｃｒｅｏ社（Ｐｒｅｐｓ（登録商標））およびＦａｔｕｋｈ等、多くの企業がこの方式を利用してページ面付け(組付け：imposition)を実行したが、これは並行印刷の実行より大幅に複雑な処理である。これらの企業は、複数の主要なベンダーが生成したポストスクリプトをページから独立したポストスクリプトに変換できただけでなく、異なるアプリケーションが生成した複数のポストスク

リプトジョブを組み合わせて１個の面付けされたポストスクリプトジョブにすることも可能であったため、更に高レベルのページ独立性が実現された。

【００２２】

同時に、印刷システムは面付けとは異なる要件を義務付ける。

ａ）印刷システムがポストスクリプト面付けソフトウェアより信頼性が高いと期待されている。面付けシステムよりもはるかに大きいポストスクリプトジョブの組を処理して、ＤＳＣおよびパターン認識方法を用いて処理できないジョブに関して充実した報告を行なうことが期待される。

ｂ）印刷システムは面付けソフトウェアよりも極めて高速でなければならない。

【００２３】

これまで見てきたように、既存のＤＳＣに基づくシステムよりも大幅に信頼性が高く且つ早い方法および装置を提供することが望ましい。

【００２４】

図５にジョブ並行方式を示す。これはページ並行方式に付随する複雑さおよび非効率性の多くを解決する。この方式では、複数のプロセッサ５５が複数のポストスクリプトジョブ５１を並行して処理する。別個に解釈および分割するための固有のオーバーヘッドが無いため、この方式は短いジョブを含む巨大な集合に対して極めて効率的である。一つジョブの印刷が終了した時点で別のジョブが処理されて印刷の準備ができている。同時に、この方式は巨大なジョブに適していない。

ａ）第１のジョブは複数のプロセッサが存在する利点を享受できない。

ｂ）ジョブプロセッサがページ記憶領域を使い切り、プリンタが先行ジョブを印刷するのを待ちながら長時間アイドル状態に陥る恐れがある。

【００２５】

この状況は、Ｃｒｅｏ社ＶＰＳまたは他のポストスクリプト方言で表された可変データ印刷（ＶＤＰ）ジョブ等の極めて長いポストスクリプトジョブにより悪化する。そのようなジョブは１個が１００，０００ページ以上を含んでいて何日間も動作することがある。この場合、ジョブ並行方式では結果的に、１個のプロセッサしか使われない一方で、残りのプロセッサをアイドリング状態に陥ってしまう。

【００２６】

ＤＳＣ準拠に戻ると、非ＤＳＣ準拠ポストスクリプトにおける主な問題は、ジョブ構造およびページ相互依存の欠落にある。

ａ）ジョブ構造の欠落とは、ポストスクリプトジョブにおける厳密且つ容易に識別可能な境界の欠如を意味する。

ｂ）ページ相互依存とは、各ページが、複数ページにわたり有効と思われる識別困難なリソースを含み得ることを意味する。

【００２７】

従って、ポストスクリプトのプロデューサが何故全てのリソースをジョブヘッダへ移さないかとの疑問が生じるだろう。その答えは、ジョブ生成が２回のパス（ジョブデータの処理）、すなわち解析パスおよび出力パスを必要とするからである。

ａ）解析を実行する間、要求されたリソースを求めてページ全体が解析される。

ｂ）出力パスを実行する間、リソースがジョブヘッダセクションに書き込まれる。それが終わらなければ独立したページを書き込むことができない。

【００２８】

アプリケーションによる高速なページ生成は、プリンタがページを処理するのと同程度に重要であり、また過去においてページ独立性がポストスクリプトプロデューサにとって必要条件ではなかったことを考えれば、ポストスクリプトジョブにおいてページが相互に依存する理由は明らかある。

【００２９】

ＰＰＭＬ等、最新のＰＤＬの導入により状況が若干変化している。ＰＰＭＬはＸＭＬベースのＶＤＰ言語であり、すなわち高速印刷を実現すべく特別に設計されたことを意味す

る。ＰＰＭＬは、多く主要な文書作成企業並びに全ての主だったプリンタコントローラメーカーが参加している標準委員会であるＰＯＤｉにより設計された。ジョブ構造に関して、ＰＰＭＬは必須ＸＭＬタグを義務付けることによりこの問題を解決している。同標準は下記を規定している。

ａ）１個のＰＰＭＬジョブは複数の文書の集合からなる。

ｂ）１個の文書の集合は複数の文書からなる。

ｃ）１個の文書は複数のページからなる。

【００３０】

ページ構造に関する限り、ＰＰＭＬはページ相互依存の問題を解決しない。ポストスクリプトページと同様に、ＰＰＭＬページは、複数ページにわたり有効と思われるリソースを含んでいてよい。これは、ＰＰＭＬページを極めて高速に出力することでデータのパス（読み込んで処理すること）を２回行うことを回避する必要に直面していた全てのＰＯＤｉメンバーの意思に基づく決定であった。その結果、ＰＰＭＬページはリソースおよびデータを以下のようにインターリーブする。

```
BeginPage
  data, resource, resource, data, data...
EndPage
```

【００３１】

ポストスクリプトとの唯一の顕著な違いは、リソースが容易に識別可能であるという点である。ＰＰＭＬジョブ構造を理解することにより、本発明の理解だけでなく既存の特許を理解しやすくなる。

【先行技術文献】

【特許文献】

【００３２】

【特許文献１】米国特許第５，６５２，７１１号明細書

【特許文献２】国際公開第０４／１０７２５９号

【特許文献３】米国特許第６，８１７，７９１号明細書

【非特許文献】

【００３３】

【非特許文献１】Ａｄｏｂｅ文書構造化規約仕様バージョン１

【非特許文献２】Ａｄｏｂｅポストスクリプト言語リファレンスマニュアル

【非特許文献３】Ａｄｏｂｅ文書構造化規約仕様

【００３４】

当分野における他の従来技術として以下がある。

１．Ａｇｆａ社、米国特許第５，６５２，７１１号（Ｖｅｎｎｅｋｅｎｓ）

２．Ｅｌｅｃｔｒｏｎｉｃ　Ｆｏｒ　Ｉｍａｇｉｎｇ社、ＷＯ出願第０４／１１０７５９号

３．Ｘｅｒｏｘ社、米国特許第６，８１７，７９１号（Ｋｌａｓｓｅｎ）

米国特許第５，６５２，７１１号はポストスクリプトを含む全てのＰＤＬに応用可能な幅広い特許である。同特許は、ＰＤＬデータストリームを並行処理する方法を記述している。これは、データコマンドおよび制御コマンドの組合せとしての印刷ジョブを定義するＰＤＬデータストリームを考慮している。データコマンドがテキスト、グラフィック、および画像のように出力装置により再生する必要があるデータを記述するのに対し、制御コマンドはデータをどのように再現するかを記述するものであり、フォント記述、ページセクション、フォームおよびオーバレイを含んでいてよい。各々の生成された独立データストリームセグメントは、単一のページまたは領域に含まれる画像を記述するデータコマンドを含み、またデータコマンドをどのように解釈すべきかを指示する制御コマンドも含んでいる。

【００３５】

ＰＤＬデータストリームは主プロセスへ送られ、ここでＰＤＬデータストリームは独立

したデータストリーム部分に分割され、独立したデータストリーム部分は複数のサブプロセスにより中間データストリーム部分に変換される。セグメントの独立性を実現するために各セグメントは、先行する全ての制御コマンドからなる当該セグメントの「変換状態」を知らなければならない。

【００３６】

本方法は、ＰＤＬストリームに関する完全な知識を必要とし、ストリームを解釈することによってのみ実現できる。解釈がボトルネックであることを理解した上で、この発明の一実施形態はこの解釈を複数のサブプロセスに分配する。変換状態の変化に遭遇したサブプロセスは全てこの変化を主プロセスに報告する。複数のサブプロセスにより生じた状態を同期させる特別な技術を用いる。

【００３７】

米国特許第５，６５２，７１１号に記述されている発明の複雑さは別として、同特許はセグメントを作成する仕組みを開示していない。例えば、ポストスクリプトの場合、「データコマンド」および「制御コマンド」が言及されていない。ほぼ全てのグラフィック演算子がインタプリタの状態を変える。残念ながら、同特許ではポストスクリプト演算子からデータ／制御コマンドへのマッピングが行なえない。

【００３８】

ＷＯ第０４／１０７２５９号明細書もまた、ポストスクリプトを含む全てのＰＤＬに適用できる幅広い特許である。その目的はページの相互依存性を克服することである。他の多くの公知技術と同様に、各ページはセグメントに分割されている。新規な点は、各々の生成されたセグメントが２個の新規ファイル、すなわち大域データファイルおよびセグメントデータファイルにより表される点である。ページをスキップするためには大域ファイルを実行する必要がある。ページを印刷するためにはセグメントデータファイルを実行する必要がある。

【００３９】

残念ながら、ＷＯ第０４／１０７２５９号明細書は、セグメントを識別する仕組みを開示していない。また、当該特許は大域データファイルおよびセグメントを構成するセグメントデータファイルを生成する仕組みも記述していない。当該特許の記述から、その本発明が「グラフィックオブジェクト」を認識および抽出可能であることを考慮し、且つＤＳＣおよびＤＳＣ関連特許を参照していない点を考慮すれば、インタプリタに基づく方式が示唆されており、従って上述のようにシステムの全体的なスループットが制限されるものと仮定することができる。

【００４０】

米国特許第６，８１７，７９１号には、ポストスクリプトジョブを独立したページに分割することが記述されている。ポストスクリプトジョブはリソース（当該特許文献の表現では、イディオム）を求めて解析され、次いでリソースが抽出されて印刷ジョブのヘッダに再配置される。次いでヘッダを各ページの先頭に置く（プレフィックスする）ことにより、当該ページに全ての必要なリソースが含まれるようにして、他のページから独立させる。各ヘッダ（ページに添付された）は当該ページに先行する全てのリソースを含んでいるが、当該ページのリソースは含んでいない。

【００４１】

同特許から分かるように、この結果、各ページに巨大ヘッダが添付される。この問題を回避すべく、米国特許第６，８１７，７９１号は「チャンク」という概念を導入し、ジョブを独立したページに分割するのではなく、ジョブを独立したチャンクに分割する。この方式では、ヘッダのオーバーヘッドはチャンク内の多数のページに均等化される（分割してならされる）。チャンクは、１ページと同程度に小さくても、またはジョブ全体と同程度に大きくてもよい。チャンク同士は独立しているため任意の順序で処理可能であり、複数の処理ノードへ分配して並行処理を行なうことができるため、これをチャンク並行性と呼ぶ。

【００４２】

チャンク並行性に関して、このチャンク並行性がどのように他の公知のチャンク並行性を用いる方式と異なるかは不明確である。例えば、既に１９９２年に公表されていた「Ａｄｏｂｅ文書構造規約仕様バージョン３」はチャンク並行性について以下のように言及している。

「例えば、ユーザは、ある文書の最初の１００のページを５台の別々のプリンタで並行して印刷すること要求する。文書マネージャはその文書を各々２０ページからなる５個のセクションに分割し、各のセクションに対して当初のプロローグおよび文書設定を複製する。」

更に、同特許は非ＤＳＣ準拠ジョブを逆に印刷する最適化された方式を示唆している。

「若干効率的な方式として、文書全体にわたり１回パス（データを読み込んで処理すること）を実行してヘッダに含まれる筈だったが含まれていない内容を見つけてヘッダに追加し、次いで当該ヘッダを１回だけ出力し、続いて全部のページを逆の順序で印刷する。」

当業者は、記述されている方式が殆ど機能しないことを知っている。その理由は、各ページが先行ページから伝播されて「ヘッダ」内では指定できない「ｓｅｔｆｏｎｔ」その他のポストスクリプト演算子を含んでいるからである、残念ながら、この最適化さていない方式は、各ページにヘッダを追加することに関連して重大な効率面での理由のため、使用できない。結論として、同特許を用いて効率的な逆印刷をどのように実行するかは明確でない。

【００４３】

しかし、米国特許第６，８１７，７９１号における主な問題は、リソースヘッダを各ページの先頭に配置する際のオーバーヘッドである。このオーバーヘッドにより、結果的に、ページ並行性を利用したテキスト処理方式の性能が最適化されない恐れがある。代替的なチャンク方式は結果的に、ロードバランシングが最適化されない（チャンクが大き過ぎる場合）か、ヘッダオーバーヘッドが膨大になる（チャンクが小さ過ぎる場合）恐れがあり、ページの複雑さ、ジョブサイズ、システム内のリソース、現在のシステム負荷、および他の要因に従い最適なチャンクサイズを推定する複雑な方法を発明する必要が生じる。

【発明の概要】

【発明が解決しようとする課題】

【００４４】

これまで見てきたように、以下を実行する方法および装置を提供することが望ましい。

１．ヘッダオーバーヘッドの累積を回避する、

２．ページ並行性を利用して上述のチャンクサイズ推定の複雑さを回避する、

３．効率的な範囲指定印刷を実現する、

４．信頼性の高い逆印刷を実現する。

本発明は、上述および他の問題を解決するものである。

【課題を解決するための手段】

【００４５】

本発明は、ページ独立性を欠くＰＤＬデータストリーム（ジョブ）の効率的な処理を行なう方法および装置を提供する。本システムは効率的に１つのジョブを、複数のページ、データ、および複数のリソースへと編成する。編成されたジョブは、以下の利点を有する。

１．編成されたジョブは当初のジョブの高レベル構造を提供する。この構造は、ジョブ解析、レポーティング、プリフライト、面付け(imposition)判定その他の処理に役立つ。

２．編成されたジョブは、効率的なページ並行処理を行なうために複数のＰＤＬプロセッサへ受け渡すことができる。

３．選択されたページまたはページ範囲を効率的に印刷することができる。

４．ページを効率的に再配置してページの逆順印刷および他のシーケンスを実現することができる。

編成されたジョブは以下の特性を有する。

１．編成されたジョブは、当初のジョブのデータおよびリソースを再配置しない。
２．編成されたジョブはワークフロー、記憶、性能その他のニーズを満たすべく、複数の形式を用いて効率的にパッケージングできる。
３．最も効率的なパッケージングは、編成されたジョブを、ポインタまたはオフセットを用いて当初のポストスクリプトジョブのセグメントを指示する、ディレクトリに似た別個の外部構造として表すことにより実現され、これにより当初のジョブが保持され、変更されたジョブを書き込むオーバーヘッドを回避することができる。

【００４６】

ポストスクリプトジョブの場合、本発明はＤＳＣ処理およびテキスト構文解析を用いる。

【図面の簡単な説明】

【００４７】

【図１】ヌルデバイスを用いたページの並行処理を示す模式図である。
【図２】ヌルデバイスを用いたページの２段階パイプライン並行処理を示す模式図である。
【図３】ディスプレイリストに基づく集中型解釈を示す模式図である。
【図４】ページ並行性を得るためのＰＤＦ方式を示す模式図である。
【図５】ジョブ並行方式を示す模式図である。
【図６】一般的な処理を示す模式図である。
【図７】リソースのリソース記憶装置への分割を示す模式図である。
【図８】オーガナイザ（編成機能）の構成要素を示す模式図である。

【発明を実施するための形態】

【００４８】

本発明の詳細な説明により当業者は、本発明を完全に発現させて実装することができる一方、実装者が可能な最高の性能を実現する際の創造性および必要なプロデューサ（製作者）の全てを最も効率的に扱える能力を限定するものではない。

【００４９】

本発明は一実施形態に関して記述されているが、本発明を当該実施形態に限定することは意図していない点を理解されたい。逆に、添付の請求項が対応する全ての代替物、変更、および等価物が包含されるものとする。

【００５０】

ポストスクリプトジョブおよびポストスクリプトに基づくＶＤＰジョブに対し複数のプロセッサを用いて可能な最高速度を得ることは複雑なタスクであり、そのためのうまい「数学的解決法」は存在しない。上述の理由により、本発明は当分野で広範な経験により検証されているいくつかの結論に基づいている。

１．ページの並行印刷はページ独立性を必要としない。ページの並行印刷は、明示的な「リソースマーキング（リソースの印付け）」と共に「ページの分離」だけを必要とする。ページディストリビュータは、ページをレンダリングするプロセッサへ当該ページ全体を受け渡すか、または当該ページに定義されたリソースだけを当該ページをレンダリングしない他のプロセッサへ受け渡す必要がある。上述の理由により、ページから独立しないように設計されているＰＰＭＬは、効率的なページ並行性の実現に理想的に適している。
２．ジョブ並行性を用いて短いジョブを最も効率的に処理することができる。短いジョブの定義はシステム、プロセッサの数、プリンタ速度、予想されるジョブの複雑さ等に依存する。いくつかのシステムにおいて、短いジョブは最大４ページまで含んでいるものと定義され、他のいくつかのシステムでは短いジョブは最大１００ページまたはそれ以上を含んでいるものと定義される。
３．リソースの集中度（濃度）は、中間サイズまたは大規模サイズのポストスクリプトジョブ内で急激に低下する。すなわち、大部分のリソースは、第１ページの前に、または第１ページ内で定義される。第２ページは通常、第１ページより少ないリソースを含んでいる。第３ページは通常、第２ページより更に少ないリソースを含んでいる。ジョブが５

００ページを含んでいる場合、２５０ページ目に何らかのリソースが含まれている可能性は低い。１００，０００を超える文書を含む典型的なポストスクリプトに基づくＶＤＰジョブの場合、最初の１００文書を過ぎて何らかのリソースが存在する可能性は極めて低い。

【００５１】

上の結論によれば、本発明の主な目的は、複数の処理ノードへ効率的に分配すべくジョブ内のページ、文書およびリソースに効率的にマーキング（印付け）を行うことによりジョブを編成(organize)することである。図６を参照するに、ページを編成する構成要素はジョブオーガナイザ６２であり、ポストスクリプトジョブ１１を受信して、編成されたジョブ６３を生成する。編成されたジョブを複数のＰＤＬプロセッサ６５に分配する構成要素はディストリビュータ６４と呼ばれている。

【００５２】

本発明の一態様は、オーガナイザがジョブを再配置する必要がなく、全てのデータおよびリソースを所定の位置に保つことができる。これが、本発明が他の発明とは異なる点であって、結果的にこれまでにない分割および並行処理の速度が得られる。実際、本発明の一実施形態において、編成されたジョブは当初（オリジナル）のジョブのセクションへの参照（ディレクトリ）リストとして表される。この言明を理解および評価するために、編成されたジョブの可能な編成およびパッケージを考慮されたい。

【００５３】

編成されたジョブは、結果的に生じる多くのセグメントとして表される。これらのセグメントは、メタデータを用いてジョブ構造を定義し、且つジョブデータを含んでいる。各セグメントはタグにより定義され、以下の７種のタグが必要である。

BeginJob
EndJob
BeginDoc
EndDoc
BeginPage
EndPage
Data

純粋なポストスクリプトジョブ（文書の表記(notion of docs)を含んでいない）の場合、次の５種のタグだけが必要である。

BeginJob
EndJob
BeginPage
EndPage
Data

２ページを含んでいる１個の文書を含む編成されたジョブの簡単な例は以下のタグを含んでいる。

```
BeginJob
  Data
  BeginDoc
    BeginPage
      Data
    EndPage
    Data
    BeginPage
      Data
    EndPage
  EndDoc
EndJob
```

編成されたジョブの正式な記述は以下の通りである。

job = BeginJob, [doc | Data]*, EndJob
doc = BeginDoc, [page | Data]*, EndDoc
page = BeginPage, [Data]*, EndPage

上の記述の言語的記述は以下の通りである。

－ジョブはＢｅｇｉｎＪｏｂおよびＥｎｄＪｏｂタグによりカプセル化されており、複数のｄｏｃおよびＤａｔａセグメントを含んでいる。

－Ｄｏｃ（またはＶＰＳ用語でいうところのブックレット（Ｂｏｏｋｌｅｔ））はＢｅｇｉｎＤｏｃおよびＥｎｄＤｏｃタグによりカプセル化されており、複数のページおよびデータセグメントを含んでいる。

－ページはＢｅｇｉｎＰａｇｅおよびＥｎｄＰａｇｅタグによりカプセル化されており、複数のＤａｔａセグメントを含んでいる。

【００５４】

ＰＰＭＬと同様に、データも明示的な範囲を含んでいてよい。範囲とは、ページ、ドキュメント、ジョブ、およびグローバルであってよい。リソースは、現在の範囲よりも高次の範囲を有するデータとして定義される。例えば、データがページ内で定義されていてジョブ範囲を有する場合、それはリソースである。リソースの従来の定義（ポストスクリプト、ＰＰＭＬ、および他のＰＤＬのリソース定義と同一）は既知とする。編成されたジョブは、文書の並行分配だけでなくページの並行分配にも適している。

【００５５】

ディストリビュータは、以下の規則に従いページの並行処理のために編成されたジョブを発行する。

－範囲がグローバル、ジョブおよびドキュメントであるデータは、当該ジョブを処理すべく指定された全てのプロセッサに分配される。

－範囲が所与ページのページであるデータは、１個のプロセッサ、すなわち当該ページを処理すべく指定されたプロセッサだけに分配される。

ディストリビュータは、以下の規則に従い文書の並行処理のために編成されたジョブを割り当てる。

－範囲がグローバルおよびジョブであるデータは、当該ジョブを処理すべく指定された全てのプロセッサに分配される。

－範囲が所与のドキュメントのドキュメントおよびページであるデータは、１個のプロセッサ、すなわち当該ドキュメントを処理すべく指定されたプロセッサだけに分配される。

編成されたジョブは、システムの記憶および性能必要を満たすようにパッケージングできる。

－編成されたジョブはＸＭＬを用いてパッケージングできる。各セグメントは、ＸＭＬ構造として表される。これは、（バイナリデータに関して知られるあらゆる問題と共に）ＰＰＭＬと同様である。

－より効率的なパッケージングは各セグメントのタグおよび長さの形式を用いる。これは公知の形式（例えばＴＡＲ形式）と同様であり、効率的なバイナリ表現を可能にする。

－更により効率的なパッケージングは、タグを含んでいてポインタまたはオフセットを用いて当初のポストスクリプトジョブのセグメントを指すディレクトリと同様の別個の外部構造として表される。これにより、本発明の一実施形態においてジョブ全体が保持されるという本発明の請求項の一つが正当化される。これは、ポストスクリプトジョブ変換の領域において公知の技術ではなく、本発明の結果としてのみ利用可能な独特の表現である。

【００５６】

いくつかの実施において、共有リソース記憶装置７５に存在する共通リソース７４の全てまたは一部を保持することがより有益であることがわかる。記憶装置は図７に示すように、オーガナイザ６２、ディストリビュータ６４、プロセッサ６６、および他のシステム

ノードの間で共有されている。

【００５７】

例えば、いくつかのシステムは共有リソース記憶装置７５に大域ＶＤＰオブジェクトを格納することに利点がある一方、他のシステムは共有リソース記憶装置７５に再使用可能な全てのＶＤＰを格納することに利点があり、また他のシステムは共通リソース記憶装置７５に全てまたは一部のポストスクリプトリソースを格納することに利点がある。そのようにする利点は、リソースを中心部に保存して、編成されたジョブのサイズを減らすことにある。いくつかのシステムでは、編成されたジョブから上述の格納されたリソースを除去すべく編成されたジョブを生成することに利点がある一方、いくつかのシステムでは、当初のジョブを指す効率的な外部構造として編成されたジョブを表現することに利点がある。いずれにせよ、編成された表現が生成された際に、本発明が当初のジョブのデータ／リソースを再配置しない点を理解することが重要である。

【００５８】

大域範囲を有するリソースに関するいくつかの考察が以下に続く。ＰＰＭＬと同様に、大域範囲を用いて、ジョブの間で大域リソースを定義して維持する。これが大域範囲の主な且つ従来の目的である。しかし、本発明の一実施形態では、大域範囲を用いて非保護ポストスクリプトジョブ、すなわちポストスクリプトインタプリタの永続的状態を変えるジョブ）を表現する。上述の分配論理を用いて、各ノードは、全てのデータを受け取る（大域範囲を有するため）。「ｓｈｏｗｐａｇｅ」演算子を無効にする（さもなければ各ノードが全部のページを印刷してしまう）ために、多くの公知の技術（ｓｈｏｗｐａｇｅの再定義、ヌルデバイスの確立等）を用いることができる。非保護ジョブを扱う当該実施形態を提示したことにより、非保護ポストスクリプトジョブを扱う本発明に依存する他の方法が可能である。

【００５９】

編成されたジョブの高いストリーミング性を認識されたい。すなわち、ページのセグメントを、それらが印付けされた（しかもページが編成される前に行なわれる場合が最も多い）直後にプロセッサに分配することができる。ジョブの編成および分配には当該ジョブに対して１回のパス（ジョブデータを読み込んで処理すること）を実行するだけで済む。

【００６０】

本発明の好適な実施形態はジョブ内のリソースを再配置せず、それらが見つかった場所に保持するにもかかわらず、リソースを再配置して、編成されたジョブ内の他の場所またはジョブの外側（図７に示すように）へ移しても本発明の趣旨が変わらない点を理解されたい。

【００６１】

例えば、本発明の一実施形態がリソースを、当該リソースが見つかったページから（美観その他の理由により）当該ページの先頭へ移すことがある。これにより当該実施形態の効率が若干低下する恐れがあるものの、ページ独立性を求めるいくつかのアプリケーションが行なう全てのリソースをヘッダに累積して当該ヘッダを各ページの先頭に置くことよりもはるかに効率的である。

【００６２】

編成されたジョブにより効率的なページスキップが可能になるため、本発明により、複数ページを並行してページ範囲処理を効率的に実行できるようになる。

【００６３】

ジョブ内のページを再配置または逆転させることはより複雑な手順である。並行ページ印刷における他の発明は、ＷＯ第０４／１０７２５９号のようにこの問題に対処しないか、または米国特許第６，８１７，７９１号のようにファイルの重要な部分で失敗するような極めて限られた解決策しか提供しない。説明のために、ページ再配置の最悪の場合である逆印刷に着目する。逆印刷は以下の技術により実現される。

１．ジョブを編成するために完全なパス（ジョブデータを読み込んで処理すること）を実行する。これにより、ページ境界およびリソースが印付けされる。

２．上述のパスを行なう間、ページ間に残るグラフィック状態に影響を及ぼすプロデューサ固有の全てのイディオムを集めて各ページに関連付ける。そのようなイディオムの例として、Ｗｉｎｄｏｗｓ（登録商標）ドライバにより生成される「ｆｏｎｔｎａｍｅＪｉ」コマンドがある。このコマンドは、フォントを「ｆｏｎｔｎａｍｅ」に設定する。これがリソースの収集および累積とは非常に異なる点に注意されたい。例えば、Ｗｉｎｄｏｗｓ（登録商標）ドライバの場合、各ページは、最後の「Ｊｉ」コマンドだけをページに関連付ければよい（リソースの場合のように先行する全ての「Ｊｉ」コマンドである必要がない）。その結果、関連付けられた状態は極めて小さくなる（通常、数百バイト以下と測定される）。

３．全てのリソースを実行する。これにより適切なポストスクリプト仮想メモリ（ＶＭ）状態が生成される。

４．ページをコマンドの処理ノードに逆順で分配する。ページを分配する前に、グラフィック状態の必要な部分を設定する小さいヘッダを追加する。

上述の方式は極めて広範な印刷ジョブで機能する。

【００６４】

本発明のこれらおよび他の目的、特徴、および利点は、本発明の例示的な実施形態を図示および記述する以下の詳細説明を添付図面と合わせて精査することにより当業者には明らかになろう。

【００６５】

オーガナイザは、ストリーミング方式で当初のジョブを構文解析して、これを解析し、非ＤＳＣ準拠性を補償して（非ＤＳＣ準拠性を打ち消して）、効率的な分配に適した良好な編成済みジョブを複数の処理ノードに出力する。多くの異なるプロデューサにより生成された多くのジョブをうまく編成するために、本発明の好適な実施形態は図８に示す構成要素を含んでいる。本発明の特性を変えることなく、これらの構成要素の名称を変更し、構成要素の役割を再配置し、構成要素を複数の下位要素に分割し、いくつかの構成要素を除去することができる。

【００６６】

構文解析は、行単位、トークン単位、および他の粒度で行なうことができるが、説明の便宜上、以下では行単位の構文解析に言及する。当初のジョブにおける各々の行は、ストリーミング方式で解析される。ある行が「％％」から始まる場合、これはＤＳＣ行の候補である。これが本当にＤＳＣ行である可能性を高めるには簡単な処理を追加すれば十分である。行がＤＳＣ行であると誤って認識されても（例えば、バイナリデータ内の行は正しいＤＳＣ行のように見える場合がある）、問題ではない。これが予想される正しいＤＳＣに合致する確率は無視できる（広範囲なテストにおいて遭遇しない）。ＤＳＣ行は重要であり、汎用ＤＳＣ処理の実行。ジョブプロデューサの識別、ジョブの構造の識別、および時にはリソースの検出にも役立つ。

【００６７】

上述のように、大部分のポストスクリプトジョブは非ＤＳＣ準拠である。しかし通常は、各々のプロデューサは、プロデューサに固有の予測可能な仕方でＤＳＣ準拠性を破る。これは、各々のプロデューサが有限プログラムであるため、限定された数の出力パターンしか生成しない可能性があるためである。効率的な並行処理のためにポストスクリプトジョブを編成するには、オーガナイザはこの非ＤＳＣ準拠性を補償する必要がある。これは、ジョブデータを解析することによりなされる。非ＤＳＣ準拠性を正しく且つ効率的に補償するために、オーガナイザはプロデューサ（「クリエイタ」としても知られる）を識別する必要がある。

【００６８】

ワードプロデューサについて若干の説明が必要である。プロデューサはＸｙｚＳｏｆｔであると言うだけでは一般に不十分である。Ｗｉｎｄｏｗｓ（登録商標）ドライバを使用するＸｙｚＳｏｆｔ、またはＬａｓｅｒＷｒｉｔｅｒドライバを使用するＸｙｚＳｏｆｔ、あるいはネイティブコード生成を使用するＸｙｚＳｏｆｔであると更に明示することが

必要である。通常、これらの出力の全ては大幅に異なっている。ＸｙｚＳｏｆｔのバージョンおよびＷｉｎｄｏｗｓ（登録商標）ドライバ等のバージョンを特定することが必要な場合もある。上述の理由により、プロデューサを識別する際にアプリケーション名、ドライバ名、バージョン等を含む完全な識別情報が必要になる。

【００６９】

このため、組み合せの数が膨大になる。この組み合せ爆発を減らす一方法は、一般に（常にではないが）、使用するドライバによらずＸｙｚＳｏｆｔパターンが同一であるという事実を利用するものである。上述の理由により、ＸｙｚＳｏｆｔパターン、Ｗｉｎｄｏｗｓ（登録商標）パターン、ＬａｓｅｒＷｒｉｔｅｒ８パターン等を別々に解析する構成要素の別々の組を有することが推奨される。そのような特定の構成要素を「プロデューサプロセッサ」と呼ぶ。（ジョブをレンダリングする複数のプロセッサと混同しないこと。）

【００７０】

プロデューサという用語に関して、アプリケーション／ドライバの組み合せについて述べる方がより正確である。プロデューサチェイン（連鎖）という用語を用いる方が更によく、これはネイティブプロデューサの異なるケースに対応できる。

－純粋なドライバ（連鎖内の要素の数は１に等しい）

－ネイティブアプリケーション（連鎖内の要素の数は１に等しい）

－アプリケーション／ドライバの組み合せ（連鎖内の要素の数は２に等しい）

－連鎖内のプロデューサの数が２より多いいくつかのケース（例：ＬａｓｅｒＷｒｉｔｅｒ８ドライバを使用するＱｕａｒｋＸＰｒｅｓｓを使用するＣｒｅｏＤａｒｗｉｎ。これは３要素プロデューサチェインを構成する）。

【００７１】

＜全体的な処理フロー＞

スナップショットにおいて、オーガナイザは行単位でポストスクリプトジョブを構文解析する。開始時点ではプロデューサチェインは空である（プロデューサは未知）。汎用ＤＳＣ処理８２が用いられる。

【００７２】

ある時点でオーガナイザ８１はプロデューサチェインの第１要素を検出する。更なる議論のため、これがＬａｓｅｒＷｒｉｔｅｒ８ドライバであると仮定する。その時点以降、各行はＬａｓｅｒＷｒｉｔｅｒ８プロセッサ（プロデューサプロセッサのインスタンス）へ送られる。

【００７３】

ＬａｓｅｒＷｒｉｔｅｒ８プロセッサは各行の高速解析を実行する。通常、関心対象でない行を除外するために行の始めおよび行の終わりで数バイトを解析すれば充分である。大多数の行はプロデューサプロセッサの関心対象外である。しかし、潜在的に行が関心対称になり得る場合、より精緻な処理が実行される。リソーススニファ（探知機能）８５により行がリソースパターンであると認識された場合、プロデューサプロセッサはプロデューサプロセッサに固有の論理を起動してリソースに印付けを行う。このプロデューサプロセッサ固有の論理には、リソースの始まりを見つけるための逆方向探索およびリソースの終わりを見つけるための順方向探索が含まれている。リソースが見つかればプロセッサはオーガナイザにリソースの開始位置および終了位置を通知する。オーガナイザは、上述のパッケージングスキームに従いリソースに印を付けて、自身の位置を当該リソースの直後まで前進させる。これにより当該リソースの処理が完了する。

【００７４】

プロデューサプロセッサが行を認識しない場合、効率的に戻る。オーガナイザは次いで、後述する汎用ＤＳＣプロセッサ論理を用いて行を処理する。

【００７５】

この方式の強みは、各プロデューサプロセッサが必要に応じて汎用ＤＳＣプロセッサのデフォルト動作を上書きできる一方、同時に汎用ＤＳＣプロセッサの能力に依存して大多

数の行を処理する点である。このように、各プロデューサプロセッサは、特定の非ＤＳＣ準拠性を補償するために必要な最も少ない数のコード行で実装することができる。より準拠度の高いプロデューサは、より簡単なプロデューサプロセッサとしてインプリメントされる。

【００７６】

引き続き上例を参照するに、オーガナイザはアプリケーションを検出する（前の段階でドライバＬａｓｅｒＷｒｉｔｅｒ８が検出された）。具体的に、これがＡｄｏｂｅＡｃｒｏｂａｔとする。オーガナイザは、これをプロデューサチェインの第２要素としてインストールする。この時点以降、オーガナイザはプロデューサチェイン内の各プロデューサプロセッサに各行を提供する。

－オーガナイザは、ＡｄｏｂｅＡｃｒｏｂａｔに行を提供する。

－当該行が拒絶された場合、オーガナイザは当該行をＬａｓｅｒＷｒｉｔｅｒ８に提供する。

－当該行が拒絶された場合、オーガナイザは汎用ＤＳＣプロセッサを用いてこれを処理する。

【００７７】

＜汎用ＤＳＣプロセッサ＞

図８において、汎用ＤＳＣプロセッサ８２は「Ａｄｏｂｅ文書構造規約仕様」に定義された汎用ＤＳＣ処理フローの役割を果たす。

【００７８】

ＤＳＣ準拠性に頼ることはできないものの、上述の「汎用処理フロー」に見られるように、汎用ＤＳＣプロセッサは極めて重要な構成要素である。オーガナイザのデフォルト動作を実行して、各プロデューサプロセッサをなるべく小さくし且つインプリメントを容易にする。汎用ＤＳＣプロセッサは、ジョブヘッダの解析、ジョブのプロローグ解析、ジョブデフォルト値の解析、リソースの解析、手続きセットの解析、ページ境界の探索、ジョブトレーラの探索、および「Ａｄｏｂｅ文書構造規約仕様」に記述された汎用ＤＳＣ処理に必要とされる他の多くの他の動作を実行する。また、並行処理用のジョブを編成するために厳密には必要とされない他のインプリメント（実装）に固有の機能を実行することがある。

【００７９】

＜クリエイタスニファ＞

クリエイタスニファ８３は、プロデューサチェインを識別する役割を果たす。上述のように、単一のクリエイタまたは単一のプロデューサだけではなく、複数のプロデューサからなる生成チェインについて議論する方がより正確である。％％ＣｒｅａｔｏｒＤＳＣを使用するのは一般に信頼性が高くない。最も信頼性の高い方式は、ＰｒｏｃＳｅｔｓ、すなわち特定のプロデューサに必要なポストスクリプトプロシージャを定義するポストスクリプトジョブ内の特別なセクションを解析することである。このように、ジョブがＬａｓｅｒＷｒｉｔｅｒ８ドライバにより生成された場合、オーガナイザはある時点でＬａｓｅｒＷｒｉｔｅｒ８ＰｒｏｃＳｅｔｓに遭遇する。ジョブがＡｄｏｂｅＡｃｒｏｂａｔアプリケーションにより生成された場合、オーガナイザはある時点でＡｄｏｂｅＡｃｒｏｂａｔＰｒｏｃＳｅｔｓに遭遇する。仮想的な例において、ＸｙｚＳｏｆｔにより生成されたが、ＸｙｚＳｏｆｔＰｒｏｃＳｅｔｓが存在しない場合、単にＸｙｚＳｏｆｔが当該ジョブで特定のＸｙｚＳｏｆｔリソースを一切使用しないことを意味するだけであり、従ってＸｙｚＳｏｆｔパターンを解析する必要がない。プロデューサのバラエティを考慮すれば、プロデューサを決定する際に％％ＣｒｅａｔｏｒＤＳＣおよび他のＤＳＣを解析することは、ある場合には依然として有益である。

【００８０】

＜ページデータスニファ＞

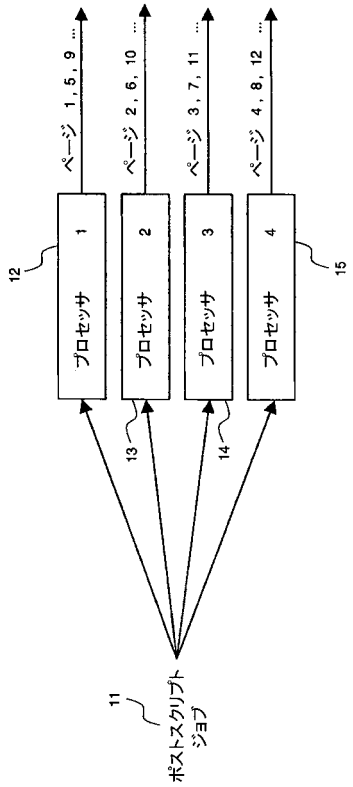
ページデータスニファ８４は、ページ全体をリソースとしてマーキングすべきか否かを決定する役割を果たす。明らかに、この論理はプロデューサごとに異なる。

【００８１】
経験的に知られているように、例えば複数のポストスクリプト面付けパッケージにおいて、所与のプロデューサに対して当該プロデューサが使用するリソースを検出および抽出するコンポーネントを常に実装することができる。多くの場合これは簡単でないことが分かっている。長時間にわたる試行錯誤が必要である。ポストスクリプト面付けアプリケーション、およびページ独立性を追求する他の方式の場合、他に現実的なオプションが存在せず、リソースを検出して抽出しなければならない。上述の理由により、そのようなアプリケーションは一般に以下の二つの方式を採用する。１）複数のプロデューサを扱うために相当な努力を払う、２）対応するプロデューサの数を制限する。
【００８２】
ページ独立性を追求しない本発明は、自由に使える他のオプションを有する。実施例が示すように、ページ上のリソースの存在を認識する方がそれらを抽出または印付けするよりも大幅に容易である。上述の理由により、本発明の実装者は、ある場合において当該ページに対する高速パス（データを読み込んで処理すること）を実行し、リソースが見つかったならばページ全体をリソースとしての印付けすることを選択できる。上で述べた「リソースの集中度はジョブ内で急激に低下する」との言明を考慮すれば、本発明のこの部分のため、極めて短時間で本発明を極めて合理的にインプリメントすることができる。明らかに、本発明のより精緻な実施形態は、上述のショートカットを控えめに用いて、最も重要なプロデューサに対してリソース印付けを実施する。
【００８３】
＜リソーススニファ＞
リソーススニファ８５は、リソースを認識および印付けする役割を果たす。リソース探知については上で述べた。実装者は、上述のリソースページのショートカットを使用しない限り、殆どの時間を製品固有のリソーススニファの実装に費やすことを覚悟しなければならない。複数の面付け実装を考慮すると、当業者は本発明を効率的に実装するために必要なリソース探知を実装することができる。
【００８４】
＜画像スニファ＞
画像スニファ８６は、画像の境界を検出して画像を効率的にスキップする役割を果たす。画像は極めて膨大になり得るため、認識して効率的にスキップすることが有益である。明らかに、ＤＳＣ規約に従い画像をスキップするために汎用ＤＳＣプロセッサ８２ロジックを用いている。このロジックは、非ＤＳＣ準拠に対応するためにプロデューサ固有のパターン認識ロジックにより拡張する必要がある。
【００８５】
＜ＥＰＳスニファ＞
ＥＰＳスニファ８７は、ポストスクリプトジョブ内のカプセル化されたポストスクリプト（ＥＰＳ）境界を検出してＥＰＳを効率的にスキップする役割を果たす。残念ながら、いくつかのプロデューサは、ＥＰＳフラグメントの埋め込みにＤＳＣ機構を使用しない。リソースの構文解析からＥＰＳの認識およびＥＰＳのスキップに失敗すれば、結果的に不正確な構文解析（例：余分なページの生成、結果的にリソース衝突を引き起こす余分なリソースの印付け）が生じる恐れがある。上述の理由により、ＥＰＳ探知のために特別なプロデューサ固有のパターン認識ロジックが必要である。
【００８６】
＜グラフィック状態スニファ＞
グラフィック状態スニファ８８は、持続的なグラフィック状態に影響を及ぼす全てのプロデューサ固有のイディオムを収集する役割を果たす。このプロデューサ固有のスニファは、ページをまたがって持続するグラフィック状態に影響を及ぼす全てのプロデューサ固有のイディオムを収集して上述のように各ページに関連付けるために必要である。そのようなイディオムの例として、ページをまたがって持続するポストスクリプト「ｓｅｔｆｏｎｔ」コマンドのエイリアスである、Ｗｉｎｄｏｗｓ（登録商標）ドライバにより生成さ

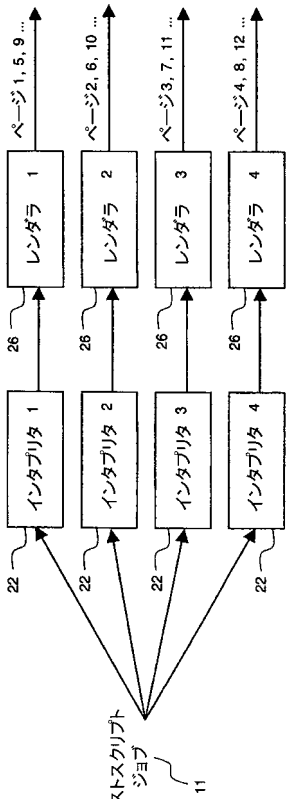
れた「ｆｏｎｔｎａｍｅＪｉ」コマンドである。
＜補遺＞
実施の形態は、以下のような側面を含んでいる。
（１）ページ独立性を欠くページ記述言語（ＰＤＬ）で記述された印刷ジョブを編成する方法であって、前記編成されたジョブはページ独立である必要がなく、複数のプロセッサにより効率的に分割および処理可能であって、
ＰＤＬジョブに対して構文解析パスを１回実行するステップと、
ＰＤＬジョブプロデューサを検出するステップと、
前記ＰＤＬジョブ内の共通リソースを検出して印付けするステップと、
前記ＰＤＬジョブ内のページ境界を検出して印付けするステップと、
オリジナルの前記ＰＤＬジョブについての前記検出を行うステップ群に従って、前記ＰＤＬジョブ内のデータおよびリソースを再配置することなく、編成された表現を生成するステップとを含む方法。
（２）ページ独立性を欠くページ記述言語（ＰＤＬ）で記述された印刷ジョブの順序替えのための方法であって、
ＰＤＬジョブに対して構文解析パスを１回実行するステップと、
ＰＤＬジョブプロデューサを検出するステップと、
前記ＰＤＬジョブ内の共通リソースを検出して印付けするステップと、
前記ＰＤＬジョブ内のページ境界を検出して印付けするステップと、
各ページのグラフィック状態を規定するコマンドを記録するステップと、
オリジナルの前記ＰＤＬジョブについての前記検出を行うステップ群に従って、前記ＰＤＬジョブ内のデータおよびリソースを再配置することなく、編成された表現を生成するステップと、
前記リソースを実行するステップと、
グラフィック状態コマンドを、順序替えして送出される前記ページ群の先頭に置くステップとを含む方法。
（３）並べ替えがページ反転である、（２）に記載の方法。
（４）前記ＰＤＬジョブがポストスクリプトジョブである、（１）に記載の方法。
（５）印付けが前記ＰＤＬジョブ内で行なわれる、（１）に記載の方法。
（６）印付けが、前記編成された表現から前記ＰＤＬジョブのセクションを指示することにより行なわれる、（１）に記載の方法。
（７）編成された表現から小さいフォームファクタが得られる、（１）に記載の方法。
（８）ページ独立性を欠くページ記述言語（ＰＤＬ）で記述された印刷ジョブの順序替えのための装置であって、
ＰＤＬジョブに対して構文解析パスを１回実行する手段と、
ＰＤＬジョブプロデューサを検出する手段と、
前記ＰＤＬジョブ内の共通リソースを検出する手段および印付けする手段と、
前記ＰＤＬジョブ内のページ境界を検出して印付けする手段と、
各ページのグラフィック状態を規定するコマンドを記録する手段と、
オリジナルの前記ＰＤＬジョブについての前記検出を行う各手段に従って、前記ＰＤＬジョブ内のデータおよびリソースを再配置することなく、編成された表現を生成する手段と、
前記リソースを実行する手段と、
グラフィック状態コマンドを前記ページ群の先頭に置く手段と、
前記ページ群を順序替えして送出する手段とを含む装置。
（９）前記ページ群を順序替えして送出する手段がページ反転の手段である、（８）に記載の装置。
（１０）前記ＰＤＬジョブがポストスクリプトジョブである、（８）に記載の装置。
（１１）マーキングが前記ＰＤＬジョブ内で行なわれる、（８）に記載の装置。
（１２）マーキングが、前記編成された表現から前記ＰＤＬジョブのセクションを指示す

ることにより行なわれる、（８）に記載の装置。
（１３）ページ独立性を欠くページ記述言語（ＰＤＬ）で記述された印刷ジョブを順序替えする装置であって、
ＰＤＬジョブに対して構文解析パスを１回実行するプロセッサと、
ＰＤＬジョブプロデューサを検出するクリエイタスニファと、
前記ＰＤＬジョブ内の共通リソースを検出および印付けするリソーススニファと、
前記ＰＤＬジョブ内のページ境界を検出して印付けするデータスニファと、
各ページのグラフィック状態を規定するコマンドを記録するプロセッサと、
オリジナルの前記ＰＤＬジョブ内の前記検出ステップに従って、前記ＰＤＬジョブ内のデータおよびリソースを再配置することなく、編成された表現を生成するプロセッサと、
前記リソースを実行するプロセッサと、
グラフィック状態コマンドを前記ページの先頭に置くプロセッサと、
前記ページを順序替えして送出するプロセッサと、
を備える装置。
【符号の説明】
【００８７】
１１　ポストスクリプトジョブ、１２　第１プロセッサ、１３　第２プロセッサ、１４　第３プロセッサ、１５　第４プロセッサ、２２　インタプリタ、２６　レンダラ、３２　集中型解釈実行プロセッサ、３３　独立ページ、３４　レンダリングプロセッサ、４２　ポストスクリプト／ＰＤＦコンバータ、４３　ＰＤＦジョブ、４４　ＰＤＦディストリビュータ、４５　複数のプロセッサ、５１　複数のポストスクリプトジョブ、５５　複数のプロセッサ、６２　ジョブオーガナイザ、６３　編成されたジョブ、６４　ディストリビュータ、６５　複数のＰＤＬプロセッサ、６６　プロセッサ、７４　共通リソース、７５　共有リソース記憶装置、８１　オーガナイザ、８２　汎用ＤＳＣプロセッサ、８３　クリエイタスニファ、８４　ページデータスニファ、８５　リソーススニファ、８６　画像スニファ、８７　ＥＰＳスニファ、８８　グラフィック状態スニファ。

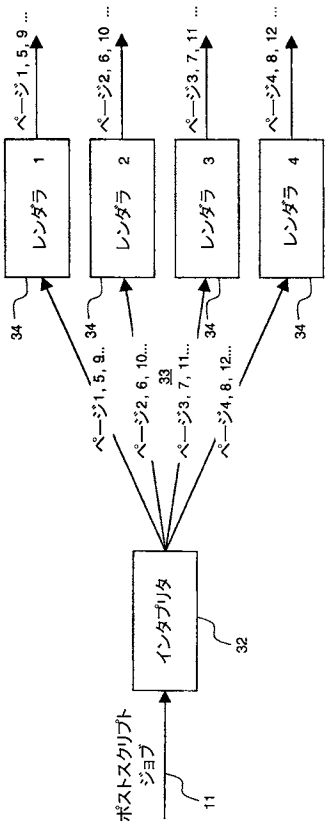
【図１】


ページ 1, 5, 9 ...
ページ 2, 6, 10 ...
ページ 3, 7, 11 ...
ページ 4, 8, 12 ...
プロセッサ 1
プロセッサ 2
プロセッサ 3
プロセッサ 4
12
13
14
15
11
ポストスクリプト ジョブ

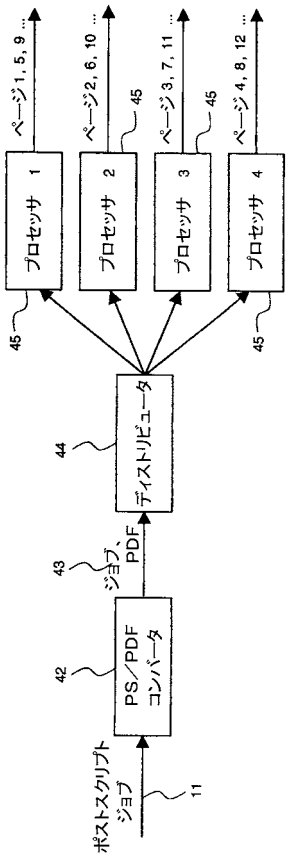
【図２】


ページ 1, 5, 9 ...
ページ2, 6, 10 ...
ページ3, 7, 11 ...
ページ4, 8, 12 ...
レンダラ 1
レンダラ 2
レンダラ 3
レンダラ 4
インタプリタ 1
インタプリタ 2
インタプリタ 3
インタプリタ 4
26
22
11
ポストスクリプト ジョブ

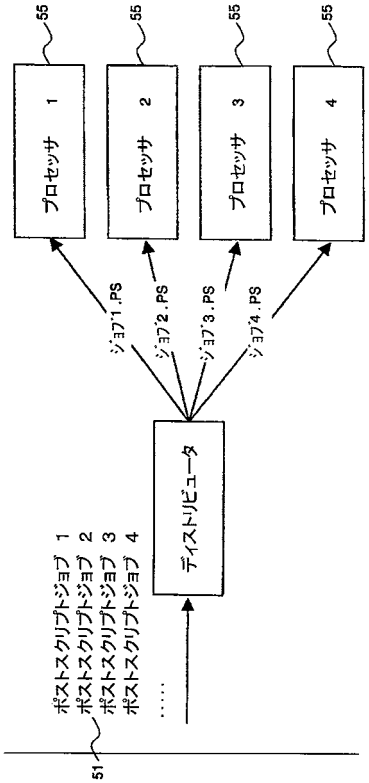
【図３】


ページ 1, 5, 9 ...
ページ2, 6, 10 ...
ページ3, 7, 11 ...
ページ4, 8, 12 ...
レンダラ 1
レンダラ 2
レンダラ 3
レンダラ 4
34
ページ1, 5, 9...
ページ2, 6, 10...
33
ページ3, 7, 11...
ページ4, 8, 12...
インタプリタ
32
11
ポストスクリプト ジョブ

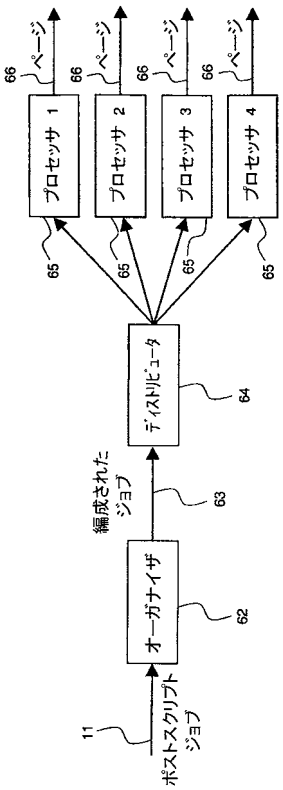
【図４】


ページ 1, 5, 9 ...
ページ2, 6, 10 ...
ページ 3, 7, 11 ...
ページ 4, 8, 12 ...
プロセッサ 1
プロセッサ 2
プロセッサ 3
プロセッサ 4
45
44
ディストリビュータ
43
ジョブ、PDF
42
PS／PDF コンバータ
11
ポストスクリプト ジョブ

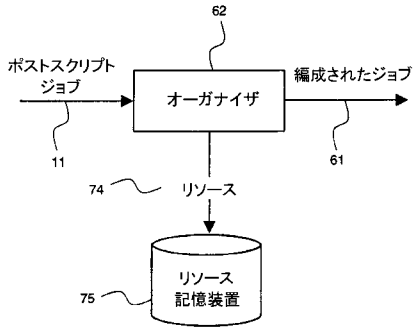
【図５】


55
プロセッサ 1
プロセッサ 2
プロセッサ 3
プロセッサ 4
ジョブ1.PS
ジョブ2.PS
ジョブ3.PS
ジョブ4.PS
ディストリビュータ
ポストスクリプトジョブ 1
ポストスクリプトジョブ 2
ポストスクリプトジョブ 3
ポストスクリプトジョブ 4
.....
51

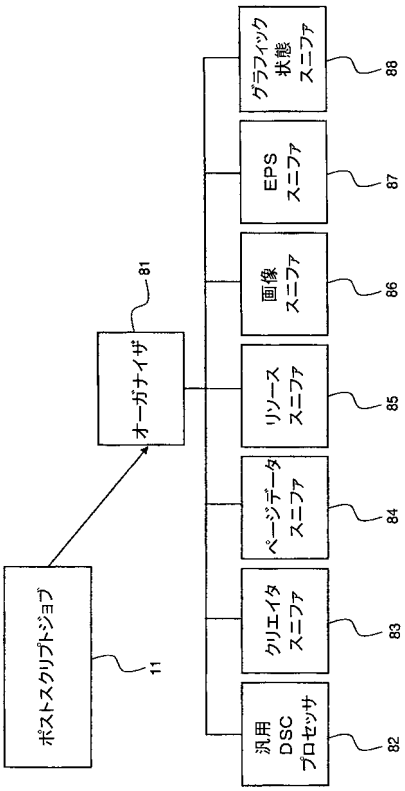
【図６】


66
ページ
プロセッサ 1
プロセッサ 2
プロセッサ 3
プロセッサ 4
65
ディストリビュータ
64
編成されたジョブ
63
オーガナイザ
62
ポストスクリプトジョブ
11

【図７】


62
ポストスクリプトジョブ
オーガナイザ
編成されたジョブ
11
61
74
リソース
リソース記憶装置
75

【図８】


ポストスクリプトジョブ
11
オーガナイザ
81
汎用DSCプロセッサ
82
クリエイタスニファ
83
ページデータスニファ
84
リソーススニファ
85
画像スニファ
86
EPSスニファ
87
グラフィック状態スニファ
88

フロントページの続き

審査官　内田　正和

(56)参考文献　米国特許出願公開第２００４／０２４３９３４（ＵＳ，Ａ１）
米国特許第０６８２５９４３（ＵＳ，Ｂ１）
特開平０８－２９７５６０（ＪＰ，Ａ）
国際公開第０４／１１０７５９（WO，Ａ１）
特開平５－２４１９３６（ＪＰ，Ａ）
特開２００６－０３９７１９（ＪＰ，Ａ）
特表２００３－５３３８３０（ＪＰ，Ａ）
特開２００４－３１０７６２（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　３／１２