(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 101802770 A
(43)申请公布日 2010.08.11

(21)申请号 200880107937.1

(22)申请日 2008.09.12

(30)优先权数据
11/858,477 2007.09.20 US

(85)PCT申请进入国家阶段日
2010.03.19

(86)PCT申请的申请数据
PCT/US2008/010658 2008.09.12

(87)PCT申请的公布数据
WO2009/038670 EN 2009.03.26

(71)申请人 伊斯曼柯达公司
地址 美国纽约州

(72)发明人 B·阿伦斯塔姆 L·海因

(74)专利代理机构 北京纪凯知识产权代理有限公司 11245
代理人 赵蓉民

(51)Int.Cl.
G06F 3/12(2006.01)
G06K 15/00(2006.01)
G06F 9/50(2006.01)

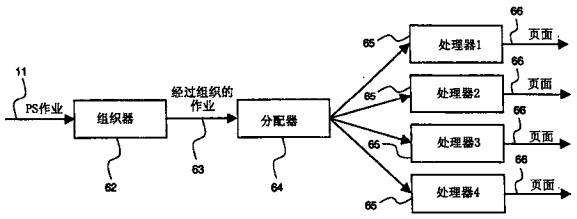
权利要求书 2 页 说明书 15 页 附图 8 页

(54)发明名称

页面描述语言的并行处理

(57)摘要

本发明描述一种高效处理缺乏页面独立性的页面描述语言(“PDL”)数据流的方法和设备。该方法和设备包括为PDL作业应用单次解析操作和由创建器监听器(83)探测PDL作业生成器。在PDL作业中共享的资源由资源监听器(85)探测。PDL作业中的页面边界由页面数据监听器(84)探测并且组织后的表示(63)被生成而无需重组PDL作业中的数据和资源。系统有效地将PDL流组织成页面、数据和资源而无需重组该流。经过组织的数据能够被有效地提交给多个PDL处理器(65)。



CN 101802770 A

1. 一种用于组织缺少页面独立性的以页面描述语言 PDL 描述的打印作业的方法，其中经过组织的作业并不要求是页面独立的并且可以被有效地分割和被多个处理器处理，所述方法包括以下步骤：

将单解析操作应用到 PDL 作业；

探测 PDL 作业生成器；

探测和标记在所述 PDL 作业中的共享资源；

探测和标记在所述 PDL 作业中的页面边界；以及

根据原始的所述 PDL 作业的所述探测步骤生成经过组织的表示，而不重组在所述 PDL 作业中的数据和资源。

2. 一种用于将缺少页面独立性的以页面描述语言 PDL 描述的打印作业的重新排序的方法，包括以下步骤：

将单解析操作应用到 PDL 作业中；

探测 PDL 作业生成器；

探测和标记在所述 PDL 作业中的共享资源；

探测和标记在所述 PDL 作业中的页面边界；

记录为每个页面定义图形状态的命令；

根据原始的所述 PDL 作业的所述探测步骤生成经过组织的表示，而不重组在所述 PDL 作业中的数据和资源；

执行所述资源；以及

将图形状态命令附加在重新排序的、将要被发送的所述页面之前。

3. 根据权利要求 2 所述的方法，其中，所述重新排序是页面逆序。

4. 根据权利要求 1 所述的方法，其中，所述 PDL 作业是 PostScript 作业。

5. 根据权利要求 1 所述的方法，其中，所述标记在所述 PDL 作业中完成。

6. 根据权利要求 1 所述的方法，其中，所述标记是通过从所述经过组织的表示指向所述 PDL 作业的部分完成。

7. 根据权利要求 1 所述的方法，其中，所述经过组织的表示导致小的形式因素。

8. 一种用于为缺少页面独立性的以页面描述语言 PDL 描述的打印作业重新排序的设备，包括：

用于将单解析操作应用到 PDL 作业的装置；

用于探测 PDL 作业生成器的装置；

用于探测和标记在所述 PDL 作业中的共享资源的装置；

用于探测和标记在所述 PDL 作业中的页面边界的装置；

用于记录为每个页面定义图形状态的命令的装置；

用于根据原始的所述 PDL 作业的所述探测步骤生成经过组织的表示，而不重组在所述 PDL 作业中的数据和资源的装置；

用于执行所述资源的装置；

用于将图形状态命令附加在所述页面之前的装置；以及

用于发送重新排序的所述页面的装置。

9. 根据权利要求 8 所述的设备，其中，用于重新排序的装置是用于使页面逆序的装置。

10. 根据权利要求 8 所述的设备，其中，所述 PDL 作业是 PostScript 作业。

11. 根据权利要求 8 所述的设备，其中，所述标记在所述 PDL 作业中完成。

12. 根据权利要求 8 所述的设备，其中，所述标记是通过从所述经过组织的表示指向所述 PDL 作业的部分完成。

13. 一种用于将缺少页面独立性的以页面描述语言 PDL 描述的打印作业的重新排序的设备，包括：

用于将单解析操作应用到 PDL 作业的处理器；

用于探测 PDL 作业生成器的创建器监听器；

用于探测和标记在所述 PDL 作业中的共享资源的资源监听器；

用于探测和标记在所述 PDL 作业中的页面边界的数据监听器；

用于记录为每个页面定义图形状态的命令的处理器；

用于根据原始的所述 PDL 作业的所述探测步骤生成经过组织的表示，而不重组在所述 PDL 作业中的数据和资源的处理器；

用于执行所述资源的处理器；

用于将图形状态命令附加在所述页面之前的处理器；以及

用于发送重新排序的所述页面的处理器。

页面描述语言的并行处理

技术领域

[0001] 本发明涉及诸如打印系统、显示系统、PDL 分析系统和 PDL 转换架构所请求的页面描述语言 (PDL) 数据的高效处理方法和设备。

背景技术

[0002] PostScript 语言是本领域普通技术人员公知的。PostScript 是一种页面描述语言 (PDL),其包含一组丰富的用于描述打印作业中的页面的命令。PostScript 和其他 PDL,诸如 IPDS、PDF、PCL 和 PPML 之间主要的不同是 PostScript 是一种编程语言。这就为表示页面内容提供了能力和灵活性,但是灵活性的代价高昂 ;在一般的 PostScript 作业中,页面并不易于解释。为了能正确地解释页面或者对 PostScript 作业进行有意义的变换就需要 PostScript 解释器。Adobe 可配置 PostScript 解释器 (CPSI) 是 PostScript 解释器的一个例子,该解释器处理 PostScript 作业并且生成位图。Adobe Distiller 是 PostScript 解释器的另外一个例子,该解释器处理 PostScript 作业并且生成 PDF 文件而不是位图。

[0003] 自从 PostScript 起始于 1984 年,全世界的工程师已经实施了众多的技术来克服 PostScript 语言的某些已知的局限性。所述局限性包括 :

[0004] a. 速度限制,该限制阻碍了以打印机额定速度执行 PostScript 作业。

[0005] b. 无法按照在多个中央处理单元 (CPU) 上并行执行页面所要求的将 PostScript 分割成单独的独立页面。

[0006] c. 无法按照高效有选择的、以页为范围地重新打印所要求的高效地打印选中的页面。

[0007] 为了理解性能问题的细节和普通实践的本质以及在下面公开的本发明,对典型的 PostScript 解释器的说明是必须的。PostScript 作业的处理由两个 (典型地为重叠的) 阶段组成 ;解释阶段和输出阶段。

[0008] —PostScript 是一种解释型语言。和任何类型的解释器 (例如 Perl 和 Java) 一样,在解释期间 PostScript 作业被解析,并且内部作业结构被创建。该内部作业结构可以是高级或低级图形对象链表 (或树)、描述作业中页面的复杂状态或任何其他专有表示格式。

[0009] 一在输出阶段期间,内部作业结构被处理,并且所请求的输出被创建。在打印系统的情况中,呈现页面并且生成光栅 (例如原始位图),并且典型地光栅被传递给打印机。在 AdobeDistiller 的情况下,生成 PDF 文件。其他格式 (诸如 AFP/IPDS) 也能够使用相似的方式被生成。

[0010] 在历史上来说,就生成的数据量而言,解释被认为是一个轻阶段,而呈现则被认为是一个重阶段。针对包括文字和图形的 PostScript 页面的典型源数据约为 100KB。若以 600×600dpi (点每英寸) CMYK 呈现,典型的原始位图页面约为 100MB,是源数据的 1000 倍。

[0011] 这就是为什么自从 PostScript 语言的起始,为了跳过呈现,工程师们曾使用“向空装置 (null-device) 写入”技术。这项技术在 Adobe 所有版本的“PostScript Language

Reference Manual(PostScript 语言参考手册)”中有描述。根据该技术,可以通过设置空装置并且接着重新建立真实装置以恢复呈现来跳过对页面的呈现。该空装置方式典型地通过再定义多个 PostScript 操作符(诸如 show(显示)、image(图像)等)来扩张以此来进一步减小解释开销。使用该空装置方式能够通过解释页面并跳过呈现来跳过页面。使用这种跳过机制,本领域普通技术人员能够实施如图 1 中所描述的页面并行处理。

[0012] 图 1 显示了 4 个处理器。在这种方法中,4 个处理器中的每一个接收完整的 PostScript 作业 11,每个处理器跳过一些页面并且处理其他的页面。例如,第一处理器 12 处理页面 1、5、9…,而第二处理器 13 将处理页面 2、6、10…,第三处理器 14 将处理页面 3、7、11…,并且第四处理器 15 将处理页面 4、8、12…。显然,这个平常的负载均衡算法可以被改进以考虑处理器的当前负载、页面的复杂性和其他特征。该负载均衡考虑可以被应用在后面所有的附图中。

[0013] 很容易看到这种方式提供的增益。假设单 CPU 系统 100 花费 100 秒处理一个完整的作业。进一步假设解释比呈现快 3 倍,这是相当合理的假设。根据这些假设,解释用时 20 秒,而呈现用时 80 秒。回到图 1,每个处理器同样花费 20 秒解释(每个处理器都需要解释整个作业),但只用 20 秒呈现(每个处理器只需要呈现页面的四分之一)。在这种情况下,处理整个作业用时 40 秒。这就获得了 2.5 倍的性能增益 (100/4 = 2.5)。

[0014] 图 2 显示了 8 个处理器。使用分开的处理器,处理过程被分割为解释和呈现。解释器 22 将经过解释的 PostScript 流发送到呈现器 26,由此实现管线并行(除了上述页面并行以外)。使用上述数字并考虑解释和呈现阶段被管线化(并行地运行),整个作业在近似 20 秒内被处理。这样就获得了 5 倍的性能增益 (100/20 = 5)。

[0015] 图 2 中所示的方法对于在早期打印时代里解释时间相比于呈现时间来说可忽略的情况是完全适当的。但是解释 / 呈现平衡从 1984 开始由于以下的因素被显著地改变了,这些因素包括 :

[0016] a) 为了加速性能,各公司通过提供极其有效的呈现系统和特有的硬件解决方案着重投资呈现技术。

[0017] b) 多 CPU 系统变得更加便宜。主流 CPU 技术的最近趋势是,现在一般的 CPU 都包括多个和独立的 CPU 一样运转的处理核。在不久的未来还可以期待 8 核、16 核和 32 核 CPU。

[0018] c) 当今更多的作业都包括非常复杂的图形和大图像,这些图形和图像要求繁重的解释。

[0019] d) 打印速度极大加快了,测量为超过 100ppm(页面每分钟),甚至超过 1000ppm。

[0020] 由于上面描述的因素,为了获得高引擎速度,对空装置的呈现,其中每个处理器解释整个 PostScript 作业,变得不合适。换句话说,解释作为固有的有序处理成为打印系统的瓶颈。例如,在图 2 中加入额外的处理器将不会提高性能,因为每个解释器需要同样用 20 秒来解释作业。

[0021] 认识到图 2 中的多个解释器重复了每个作业,本领域普通技术人员能够将解释孤立开并且将其转移到图 3 所示的单独的处理器中。在该图中,中央解释处理器 32 解释作业 PostScript 11,并且生成某包含独立页面 33 的内部作业结构(显示列表)。该独立页面 33 的显示列表被发送到单个的呈现处理器 34。这种方法的主要优势是,相比于图 2 所描述的方法只需要 5 个 CPU 就能获得相同的性能。此外,使用更强大的 CPU 用于中央解释器处

理器 32 能够稍微减小一点解释瓶颈。

[0022] 这种方法的一个严重缺点是它的复杂度，将 PostScript 处理器分成运行在分开的结点上的独立解释器和呈现器是一个复杂的过程。其需要显著地改变代码并且需要使用源代码来进行这种改变。这种方法的主要缺陷是解释器仍然是瓶颈。使用上面的例子中建议的数字，增加呈现器处理器 34 的数量并不会提高性能。

[0023] 考虑到上面的描述，期望提供一种移除作为瓶颈的解释器的方法和设备，从而提高系统的整体速度。还期望提供不要求更改解释器的方法和设备。

[0024] 对中央解释方法的已知变化是如图 4 所示的 PDF 方法。在该方法中，PS 到 PDF 的转换器 42 将 PostScript(PS) 作业 11 转换成 PDF。所创建的 PDF 43 被 PDF 分配器 44 分配到多个处理器 45。可以使用大量可用的实用程序将 PostScript 转换成 PDF。Adobe Distiller 可能是其中最著名的一种。

[0025] 也有大量的将 PDF 作业 43 分配到处理器 45 的方法，包括 ：

[0026] a) 将整个 PDF 文件发送到所有的处理器。每个处理器被告知呈现哪些页面。

[0027] b)PDF 作业可以被转换成一系列单页面的 PDF 文件。这些单页面的 PDF 文件能够被发送到处理器，从而每个处理器只接收被要求的 PDF 页面来呈现。

[0028] c)PDF 可以被转换成一系列单页面的 PostScript 文件。这些单页面的 PostScript 文件能够被发送到处理器，从而每个处理器只接收被要求的 PostScript 页面来呈现。

[0029] d) 取代单 PDF 或 PostScript 页面，可以生成页面块并且页面块可以被分配到所要求的处理器，从而降低单页面的潜在资源开销。作为变化，整个作业能够被分割为与处理器数量相等数量的块（在本例子中为 4 块）。

[0030] 这些 PDF 方法是可行的处理并且在行业内是已知的。同时，它们都具有和上面讨论过的“中央解释”方法相同的主要缺陷 ；由于 PS-PDF 转换器是 PostScript 解释器，所以转换器就成为瓶颈。此外，已知转换成 PDF 会为对转换器增加额外的大量开销，从而产生一个甚至更大的瓶颈。该瓶颈阻碍了通过增加额外的处理器来扩展系统。

[0031] 回到图 4，取代将 PostScript 转换成 PDF，采取其他的转换是可能的。例如，PostScript 转换成 PostScript、PostScript 转换成 AFP 或 PostScript 转换成 XPS。但是由于所有这些转换都是 PostScript 解释器的实例，所以它们都具有和上面讨论过的“中央解释”方法和“PDF 方法”相同的主要缺点 ；转换器变成瓶颈，从而阻碍了通过增加额外的处理器来扩展系统。

[0032] 考虑到上面的描述，期望提供将作为瓶颈的解释器移除的方法和设备，从而提高系统的整体速度。此外期望提供避免将 PostScript 转换成 PDF 或其他语言的方法和设备。

[0033] 认识到和 PostScript 作业的无组织本质相关的这些问题，Adobe 早在 1986 就出版了“Adobe Document Structuring ConventionsSpecification Version 1”（文档结构转换规范第一版，DSC 规范）。最著名的 DSC 规范第三版在 1992 年出版。在该规范中有一个独立的部分名为“并行打印”。在该部分中说明页面并行打印是符合 DSC 的 PostScript 作业的目的之一。

[0034] DSC 规范定义了一组标签 (tag)，其允许对 PS 资源简单的解析和重组页面。此外，它要求如果生成器输出“％！ PS-Adobe-3.0”，就保证该 PostScript 文件是符合 DSC 的。遗憾的是，事实是几乎所有的主要 PostScript 生成器都插入“％！ PS-Adobe-3.0”，而这些文

件很少是符合 DSC 的。

[0035] 然而，在实践中显示，通过解析 DSC 注释和针对生成器的模式可以成功地将大的 PostScript 作业组分割成独立的页面。尽管这种处理是相当复杂的，但是自从 1988 开始多家公司都已经成功地运用了这种方法。例如，多家公司，诸如 Creo(Preps ®) 和 Farukh 采用这种方法进行拼版，这是比实现并行打印明显更加复杂的过程。这些公司不仅能够将由多个主要供应商产生的 PostScript 转变成页面独立的 PostScript，他们也能够将由不同的应用程序生成的多个 PostScript 作业组合成一个拼版的 PostScript 作业，从而获得更高水平的页面独立性。

[0036] 同时，打印系统有和拼版不同的要求 ：

[0037] a) 期望打印系统比 PostScript 拼版软件更可靠。期望打印系统处理比拼版系统大得多的 PostScript 作业组并且适度报告关于那些它使用 DSC 和模式识别方法不能处理的作业。

[0038] b) 打印系统需要比拼版软件快很多。

[0039] 考虑到上面的描述，期望提供比现有基于 DSC 的系统显著地更可靠更快的方法和系统。

[0040] 图 5 显示了一种作业并行方法。该方法致力于页面并行方法的许多复杂性和低效率。在该方法中，多个处理器 55 并行地处理多个 PostScript 作业 51。由于分开的解释和分割没有固有的开销，该方法对于大的短作业组是非常高效的 ；当一个作业完成打印时另一个作业被处理并且准备被打印。同时，这种方法不适合于大作业 ：

[0041] a) 第一个作业并不受益于多处理器。

[0042] b) 作业处理器可能在长时间内用完页面存储器并空闲，等待打印机打印上一个作业。

[0043] 对于非常长的 PostScript 作业，诸如在 Creo VPS 和其他 PostScript 语言中表述的可变数据印刷 (VDP) 作业，情况会恶化。一个这种长作业包含超过 100,000 页并且运行好几天。在这种情况下，作业并行方法将明确地导致只利用到一个处理器而剩下的处理器都保持空闲。

[0044] 回到 DSC 符合性，非 DSC 符合的 PostScript 的主要问题是缺乏作业结构和页面独立性。

[0045] a) 缺乏作业结构意味着 PostScript 作业中的页面之间缺少精确的和易于识别的边界。

[0046] b) 页面独立性意味着每个页面都可能包含难以识别的资源，预计这些资源在页面范围之外维持。

[0047] 因此，可能会有疑问，为什么 PostScript 生成器不将所有的资源都移到作业头中。答案是因为作业的产生需要两个操作，分析操作和输出操作。

[0048] a) 在分析过程中，为所请求的资源分析所有的页面。

[0049] b) 在输出操作过程中，资源被写入到作业头部分。只有在此之后才对独立的页面写入。

[0050] 因为应用程序高速生成页面是和打印机消耗页面同样重要的，并且考虑到在过去 PostScript 生成器不要求页面独立性，这就明确了在 PostScript 作业中的页面互相依赖

的原因。

[0051] 随着引入最新的 PDL（诸如 PPML），情况稍微有所改变。PPML 是基于 XML 的 VDP 语言，这意味着该语言是为了获得高打印速度而特别设计的。PPML 是标准委员会 PODi 设计的，该委员会包括所有主要的打印控制器生产商以及许多主要的文档生成公司。关于作业结构，PPML 通过要求强制 XML 标签来解决这个问题。该标准指出：

[0052] a)PPML 作业由文档组组成

[0053] b) 文档组由多个文档组成

[0054] c) 文档由多个页面组成

[0055] 当涉及到页面结构时，PPML 不解决页面互相依赖的问题。和 PostScript 页面一样，PPML 页面可能包含期望在页范围以外维持的资源。这是所有 PODi 成员做出的有意识的决定，其指出需要以非常高的速度输出 PPML 页面，从而避免两次经过数据。作为结果，PPML 页面交插资源和数据，如同是：

[0056] BeginPage

[0057] data，resource，resource，data，data…（数据，资源，资源，数据）

[0058] EndPage

[0059] 该 PPML 与 PostScript 的唯一的主要区别是，资源是容易地可识别的。对 PPML 作业结构的理解将有助于理解现有的模式，并且理解本发明。

[0060] 本领域中其他的现有技术包括：

[0061] 1.Agfa，第 5,652,711 号美国专利 (Vennekens)。

[0062] 2.Electronics For Imaging，WO 04/110759 申请。

[0063] 3.Xerox，第 6,817,791 号美国专利 (Klassen).

[0064] 第 5,652,711 号美国专利是宽专利，适用于包括 PostScript 的所有 PDL。该专利描述了用于并行处理 PDL 数据流的方法。该方法考虑了将打印作业定义为数据命令和控制命令的组合的 PDL 数据流。数据命令描述必须由输出装置再生的数据，例如文字、图形和图像，而控制命令描述数据如何被再生，并且可包括字体描述，页面部分、格式和覆盖。每个生成的独立数据流段都包括数据命令，以描述包括在单个页面或区域中的图像，并且也包括控制命令，以指示数据命令该如何被解释。

[0065] PDL 数据流被递交到主处理程序，该主处理程序将 PDL 数据流分割成独立的数据流段，这些数据流段通过多个子处理转换成中间数据流部分。为了获取段独立性，每个段必须知道针对每个段的“翻译状态”，该“翻译状态”由所有先前的控制命令组成。

[0066] 该方法要求对 PDL 流的完整了解，这只能通过解释该流来实现。由于认识到解释是瓶颈，本发明的实施例之一将这个解释分配给多个子处理。遭遇翻译状态变化的任意子处理都将这种变化报告给主处理。使用特定的技术来同步由多个子处理创建的状态。

[0067] 除了在第 5,652,711 号美国专利中描述的发明的复杂性以外，该专利也没有公开创建段的机制。例如，在 PostScript 的情况下，没有“数据命令”和“控制命令”的概念。几乎所有的图形操作符都改变解释器的状态。遗憾的是，该专利没有提供从 PostScript 操作符到数据 / 控制命令的映射。

[0068] WO 04/110759 同样也是宽专利，可应用到包括 PostScript 的所有 PDL。该专利的目的是克服页面的相互依赖性。如同许多其他已知的技术那样，每个页面都被分割成段。这

里的新颖之处是，每个生成的段都由两个新文件表示：全局数据文件和段数据文件。为了跳过页面，需要执行全局文件。为了打印页面，需要执行段数据文件。

[0069] 遗憾的是，WO 04/110759 没有公开识别段的机制。该专利也没有描述创建全局数据文件和组成段的段数据文件的机制。从该专利的描述中，考虑到该发明能够识别和提取“图形对象”并且考虑到没有参考 DSC 和 DSC 相关专利，可以假设暗示了一种基于解释器的方法，从而，如上所讨论的，限制了系统的整体处理能力。

[0070] 第 6,817,791 号美国专利描述了将 PostScript 作业分割成独立页面。针对资源（习惯用语，根据该专利的语言）分析该 PostScript 作业；接着提取资源并且在打印作业头中重组资源。接着该头被置于每个页面之前，从而使得该页面包含所有必要的资源，从而使该页面独立于其他的页面。每个头（被附在页面上）包含该页面之前的所有资源，但是不包括该页面的资源。

[0071] 如同该专利所承认的，其结果是页面被附有巨大的头。为了巧妙解决这个问题，第 6,817,791 号美国专利引入了“块”的概念；取代将作业分割为独立的页面，该作业被分割为独立的块。在这种方法中，头开销被块中的多个页面分摊。块可以小到一个页面或大到整个作业。由于块是独立的，所以它们可以以任何顺序被处理，并且可以被分配给多个处理结点进行并行处理，从而被称为块并行。

[0072] 关于块并行，并不清楚该块并行与其他已知的块并行方法的区别。例如，早在 1992 年出版的“Adobe 文档结构转换规范第三版 (AdobeDocument Structuring Conventions Specification Version 3)”提到了块并行：

[0073] “例如，用户请求文档的前 100 页在 5 个单独的打印机上并行地打印。文档管理器将该文档分割成 5 个部分，每部分 20 页，为每个部分复制原始序言和文档设置。”

[0074] 此外，该专利建议一种逆序打印非 DSC 符合的作业的优化方法。

[0075] “一种略微更加高效的方法是单次穿过文档，寻找材料，该材料本应当在头中，但却不在，并且将该材料附到头上并且接着只输出该头一次，其后跟着相反顺序的所有页面。”

[0076] 本领域一般技术人员知道所描述的方法很少起作用。这是因为每个页面可能会包含“字体设置 (setfont)”和其他的 PostScript 操作符，其他的 PostScript 操作符源自前面的页面并且不能在“头”中被指定。遗憾的是，因为和将头加到每个页面相关的严重效率原因，该未优化的方法不能被使用。结论是，不明确如何应用使用本专利实施有效的逆序打印。

[0077] 然而，第 6,817,791 号美国专利主要的问题是，将资源头附加在每个页面之前所造成的开销。该开销会导致使用页面平行的文字处理方法的不是最理想的性能。替代的块方法将导致不是最理想的负载平衡（如果块太大）或头的大开销（如果块太小）和为了根据页面复杂度、作业大小、系统中的资源、当前系统负载和其他的因素估计理想的块的大小而需要发明复杂的方案。

[0078] 考虑到上面的描述，期望提供一种方法和设备，该方法和设备：

[0079] 1. 避免累积的头开销，

[0080] 2. 使用页面并行，从而避免上面提及的块大小的估计复杂性，

[0081] 3. 获得有效的范围打印，以及

[0082] 4. 获得可靠的页面逆序。

[0083] 本发明克服了上面指出的问题以及其他的问题。

发明内容

[0084] 本发明提供了一种高效处理缺乏页面独立性的 PDL 数据流（作业）的方法和设备。系统将作业有效地组织成页面、数据和资源。经过组织的作业具有以下益处：

[0085] 1. 经过组织的作业提供原始作业的高层次结构。该结构是用于作业分析、报告、输出前的准备、拼版决定和其他处理的手段。

[0086] 2. 经过组织的作业可以被传递给多 PDL 处理器，用于高效的页面并行处理。

[0087] 3. 可选的页面或页面范围可以被高效地打印。

[0088] 4. 页面可以被高效地重组以实现逆页序打印或以其他顺序打印。

[0089] 经过组织的作业具有下列特性：

[0090] 1. 经过组织的作业不重组原始作业的数据和资源。

[0091] 2. 经过组织的作业能够使用多种格式被有效地包装以满足工作流程、存储、性能和其他的需求。

[0092] 3. 能够通过将经过组织的作业表达为分离的外部结构来实现最有效的包装，其与目录相似，使用指针或偏移值指向原始 PostScript 作业的段，以此保护原始作业并且避免写入经过修改的作业的开销。

[0093] 就 PostScript 作业而言，本发明使用 DSC 处理和文字解析。

附图说明

[0094] 图 1 是示意图，图解说明了使用空装置的页面并行处理。

[0095] 图 2 是示意图，图解说明了使用空装置的页面二级管线式并行处理。

[0096] 图 3 是示意图，图解说明了基于显示列表的中央解释。

[0097] 图 4 是示意图，图解说明了用于页面并行的 PDF 方法。

[0098] 图 5 是示意图，图解说明了作业并行方法。

[0099] 图 6 是示意图，图解说明了一般处理图。

[0100] 图 7 是示意图，图解说明了将资源分割到资源存储器。

[0101] 图 8 是示意图，图解说明了组织器部件。

具体实施方式

[0102] 本发明的详细描述将允许本领域普通技术人员实施完整表述的本发明，同时不限制实施者在获取最佳的可能性能时的创造性和最有效地操纵所有要求的生成器的能力。

[0103] 虽然本发明是结合实施例之一描述的，但是应当理解的是，其并不试图将本发明限制到该实施例。相反地，其意图涵盖所附权利要求所涵盖的所有的替代、修改和等价物。

[0104] 使用多处理器为 PostScript 作业和基于 PostScript 的 VDP 作业获得最高可能的速度是一项复杂的工作并且对于其没有好的“数学解决方案”。这就是为什么本发明是基于本领域的广泛经验证明的一些结论的原因，这些结论包括：

[0105] 1. 页面并行打印不要求页面独立性。页面并行打印只要求具有明确“资源标记”

的“页面分离”。页面分配器需要或者将整个页面发送到呈现该页面的处理器或者只将在该页面中定义的资源发送到不呈现该页面的其他处理器。这就是为什么设计时不是页面独立的 PPML 理想地适合于获得高效的页面并行的原因。

[0106] 2. 使用作业并行能够最高效地处理短作业。短作业的定义依赖于系统：处理器数量、打印机速度、期望的作业复杂度等等。对于一些系统，短作业被定义为具有不多于 4 个页面，而对于另一些系统，短作业可以被定义为具有不多于 100 个页面甚至更多。

[0107] 3. 在中等大小或大的 PostScript 作业中的资源集中度快速地下降。也就是说，大部分资源都在第一页之前或在第一页内定义。第二页典型地包含少于第一页的资源。第三页所包含的资源典型地更少于第二页。在作业包含 500 个页面的情况下，第 250 页不太可能包含任何资源。对于包含多于 100,000 个文档的基于 PostScript 的典型 VDP 作业，在其前 100 个文档之后的文档几乎不可能包含任何资源。

[0108] 根据上述结论，本发明的主要目标是通过有效地标记作业中的页面、文档和资源来组织作业，用于有效地分配到多个处理结点。参考图 6，组织页面的部件是作业组织器 62，该组织器接收 PostScript 作业 11 并且生成经过组织的作业 63。将经过组织的作业分配到多个 PDL 处理器 65 的部件被称作分配器 64。

[0109] 本发明的一个方面是，组织器不需要重组作业，它可以原地保持所有的数据和资源。这就是发明和其他发明不同的地方，并且也产生了空前的分割和并行处理的速度。事实上，在本发明的一个实施例中，经过组织的作业被表示为对原始作业的部分的参考（目录）清单。为了理解和体会该语句，考虑可能的组织和对经过组织的作业的包装。

[0110] 经过组织的作业被表示为许多有序段。这些段使用元数据定义作业结构，并且包含作业数据。每个段都用标签定义，下面就是所需的 7 个标签：

[0111] -BeginJob

[0112] -EndJob

[0113] -BeginDoc

[0114] -EndDoc

[0115] -BeginPage

[0116] -EndPage

[0117] -Data

[0118] 对于纯 PostScript 作业（其不具有文档概念）只需要下列 5 个标签：

[0119] -BeginJob

[0120] -EndJob

[0121] -BeginPage

[0122] -EndPage

[0123] -Data

[0124] 包含具有两个页面的一个文档的简单组织的作业的例子包括下列标签：

[0125] BeginJob

[0126] Data

[0127] BeginDoc

[0128] BeginPage

[0129] Data

[0130] EndPage

[0131] Data

[0132] BeginPage

[0133] Data

[0134] EndPage

[0135] EndDoc

[0136] EndJob

[0137] 经过组织的作业的正式描述为：

[0138] 作业＝ BeginJob，[doc|Data]*，EndJob

[0139] 文档＝ BeginDoc，[page|Data]*，EndDoc

[0140] 页面＝ BeginPage，[Data]*，EndPage

[0141] 上述正式描述的口头描述为：

[0142] ——作业被标签 BeginJob 和 EndJob 封装并且包含多个文档 (doc) 段和数据 (Data) 段。

[0143] ——文档（或 VPS 术语中的目录单 (booklet)）被标签 BeginDoc 和 EndDoc 封装并且包含多个页面 (page) 段和数据 (data) 段。

[0144] ——页面被标签 BeginPage 和 EndPage 封装并且包含多个数据 (Data) 段。

[0145] 与 PPML 相似，数据可以包含明确的作用域。该作用域可以是：页、文档、作业和全局。资源被定义为具有比当前作用域更高作用域的数据。例如，如果数据被定义在页面中且具有作业作用域，那么该数据就是资源。读者将理解这种常规的资源定义（同在 PostScript、PPML 和其他 PDL 中的资源定义相同）。经过组织的作业适合于页面并行分配和文档并行分配。

[0146] 根据如下所述规则，分配器分派经过组织的作业，用于页面并行处理：

[0147] ——作用域＝全局、作业和文档的数据被分配给所有被指派处理该作业的处理器。

[0148] ——作用域＝页面的给定页面的数据只被分配给一个处理器——被指派处理该页面的处理器。

[0149] 根据如下规则，分配器将经过组织的作业分派用于文档并行处理：

[0150] ——作用域＝全局和作业的数据被分配给所有被指派处理该作业的处理器。

[0151] ——作用域＝文档和给定文档的页面的数据只被分配给一个处理器——被指派处理该文档的处理器。

[0152] 经过组织的作业能够被包装以满足该系统的存储和性能需求：

[0153] ——经过组织的作业可以使用 XML 包装。每个段都被表示为一个 XML 构造。这与 PPML 相似（具有所有关于二进制数据的已知问题）。

[0154] ——一种更加高效的包装是为每个段使用标签和长度格式。这与已知的格式相似，例如 TAR 格式，并且允许高效的二进制表示。

[0155] ——更加高效的包装被表示为分离的外部结构，其与包含标签和使用指针或偏移值指向原始 PostScript 作业的段的目录相似。这就证明了本发明的权利要求之一，在本发

明的实施例之一中整个作业被保存。这是一种 PostScript 作业变换领域不公知的独特的表示方式，并且只能作为本发明的结果获得。

[0156] 一些实施发现保持所有的或一些共同资源 74 存留在共享的资源存储器 75 中是更加有益的。如在图 7 中所描写的，该存储器被共享在组织器 62、分配器 64、处理器 66 和其他系统结点之间。

[0157] 举例来说，一些系统可能受益于在共享的资源存储器 75 中存储全局 VDP 对象，而其他的系统可能受益于在共享的资源存储器 75 中存储所有可再使用的 VDP 对象，并且另外一些系统可能受益于在共享的资源存储器 75 中存储所有或一些 PostScript 资源。这样做的益处是在中心位置保存资源并且减少经过组织的作业的尺寸。一些系统可能受益于创建经过组织的作业，该作业将上述存储的资源从经过组织的作业中移除，而一些系统可能受益于将经过组织的作业表示为指回原始作业的有效外部结构。在任一情况下，理解在产生经过组织的表示时本发明没有重组原始作业的数据 / 资源是重要的。

[0158] 下面是关于全局作用域资源的一些考虑。与 PPML 相似地，全局作用域被用于定义和保存作业之间的全局资源。这是全局作用域的主要和常规的目的。但是在本发明的实施例之一中，全局作用域被用于表示不受保护的 PostScript 作业，这些作业改变 PostScript 解释器的永久状态。使用上面描述的分配逻辑，每个结点将接收所有的数据（因为该数据具有全局作用域）。为了消除“显示页面 (showpage)”操作符的效应（否则将导致在每个结点打印所有的页面），可以使用许多已知的技术（重新定义“显示页面”、建立空装置和其他更多的技术）。给出处理不受保护的作业的这个实施例，依赖本发明的用于处理不受保护的 PostScript 作业的其他方法是可行的。

[0159] 读者必须意识到是经过组织的作业的高流动本质。这就是说，页面的段可以在他们被标记（该过程的发生通常甚至在页面被组织之前）后立即被分配给处理器。只要求通过作业一次来组织和分配该作业。

[0160] 尽管本发明的优选实施例并不重组作业中的资源并且将这些资源保存在它们被发现的地方，但是应当理解的是重组资源和将它们移到经过组织的作业的其他位置或者甚至是作业之外（如图 7 所示）并不改变本发明的精神。

[0161] 举例来说，本发明的一个实施例可以将资源从发现该资源处的页面中移到该页面的开头（为了美化或其他的理由）。尽管这很有可能使得该实施例不那么高效，但其仍然比将所有的资源都堆积到头并且将该头附加在每个页面之前（这是一些寻求页面独立性的应用程序已经这样做的）要显著地高效。

[0162] 由于经过组织的作业允许高效的页面跳跃，所以本发明允许高效的页面并行的以页为范围的处理。

[0163] 在作业中重组或使页面逆序是一个更加复杂的过程。在并行页面打印领域的其他发明或者并不致力于这个问题，例如 WO 04/110759，或者给出了非常有限的解决方案，例如美国专利 6,817,791，该解决方案在大部分文件上失败。为了讨论，我们专注于作为页面重组的最糟情况的逆序打印。该逆序打印是通过下列技术实现的：

[0164] 1. 运行组织作业的完整操作。这将会标记页面边界和资源。

[0165] 2. 在上述操作期间收集所有的针对生成器的习惯用语，这些习惯用语影响持续在页面之间的图形状态并且将该图形状态同每个页面相关联。这种习惯用语的一个例子

是由 Windows 驱动器生成的“fontname Ji”（字体名称 Ji）命令。该命令将字体设置到‘fontname’。应当注意到，这与收集资源和堆积资源有着很大的区别：例如，在 Windows 驱动器下，每个页面需要仅仅将最后的‘Ji’命令与页面关联（不是将所有之前的‘Ji’命令都与资源关联）。结果是，这种关联状态是极其小的（经测量典型地为几百字节或者甚至更小）。

[0166] 3. 执行所有的资源。这就创建了适当的 PostScript 虚拟存储 (VM) 状态。

[0167] 4. 以逆顺序将页面分配到处理结点。在分配页面之前添加微小的头，该头设置所需的图形状态部分。

[0168] 上述方法将对宽广范围的打印作业起作用。

[0169] 本发明的这些和其他目标、特征和优点对于本领域技术人员来说在结合显示和说明了本发明示例实施例的附图阅读了下列详细的描述之后将变得明显。

[0170] 组织器部件以流方式解析原始作业、分析该原始作业、补偿非 DSC 符合性并且输出良好形成的经过组织的作业，该作业适合高效地分配到多个处理结点。为了成功地组织由大量不同的生成器生成的大量作业，本发明的一个优选实施例包含显示在图 8 中的部件。这些部件可以被重命名并且部件责任可以被重新安排，这些部件被分成多个子部件，甚至移除其中一些部件而不会改变本发明的本质。

[0171] 可以逐行地、逐个标记地和以其他的粒度进行解析，但是为了方便描述而参考按行解析。原始作业中的每一行以流的方式被分析。如果行以“％％”开头，那么其为候选的 DSC 行。一些简单的额外处理足够增加其确实为 DSC 行的可能性。如果一行被错误地识别为 DSC 行（例如二进制数据中的行可能会看上去像是有效的 DSC 行），这也不是问题；该行与任何有效的、期望的 DSC 匹配的概率是可以忽略不计的（在大量的测试中未遇到）。DSC 行是重要的并且其有助于执行一般的 DSC 处理，以识别本作业的创建器，识别作业的结构并且有时甚至探测资源。

[0172] 如上所述，大多数 PostScript 作业是非 DSC 符合的。但是典型地，每个生成器都以针对生成器的、可预测的方式破坏 DSC 符合性。这是因为每个生成器作为有限程序可能只生成数量有限的输出模式。为了组织 PostScript 作业以进行高效的并行处理，组织器需要补偿这种非 DSC 符合性。该补偿是通过分析作业数据来完成的。为了为非 DSC 符合性获得正确高效的补偿，组织器需要识别生成器（也被称为“创建器”）。

[0173] 需要对字生成器进行一些说明。假设生成器是 XyzSoft 通常是不够的。需要进一步地阐明该生成器是使用 Windows 驱动器的 XyzSoft 或者是使用 LaserWriter 驱动器的 XyzSoft 或者是使用本机代码生成的 XyzSoft。所有这些输出通常是非常不同的。有时甚至需要指定 XyzSoft 的版本和 Windows 驱动器的版本等。这就是为什么生成器需要由完整的身份集识别，这种身份集包括应用程序名称、驱动器名称和版本等等。

[0174] 这可能会生成大量的组合。一种减少这种组合激增的方法是支持以下事实：通常（但不是总是）情况下，XyzSoft 模式是相同的、不依赖于所使用的驱动器的。这就是为什么建议使用单独的一组部件来分别地分析 XyzSoft 模式、Windows 模式、LaserWriter8 模式等。我们将这种特有的部件称作“生成器—处理器”。（不要与呈现作业的多个处理器混淆。）

[0175] 就生成器术语而言，更加精确的是讨论应用程序 / 驱动器的组合。更好的是使用

术语生成器链，这适应于不同情况下的本机生成器：

[0176] ——纯驱动器（链中的元件数量为 1）

[0177] ——本机应用程序（链中的元件的数量为 1）

[0178] ——应用程序 / 驱动器组合（链中的元件数量为 2）

[0179] ——在链中的生成器的数量大于 2 的一些情况下（例如应用使用 LaserWriter8 驱动器的 QuarkXPress 的 Creo Darwin，就创建了 3 元件生成器链）。

[0180] 一般处理流程

[0181] 在抽点打印中，组织器逐行地解析 PostScript 作业。开始时生成器链是空的（生成器未知）。使用一般的 DSC 处理 82。

[0182] 在某个时刻，组织器 81 探测在生成器链中的第一元件。为了进一步讨论，我们假设该第一元件为 LaserWriter8 驱动器。从那时刻起，每行都被提交给 LaserWriter8 处理器（生成器—处理器的实例）。

[0183] LaserWriter8 处理器对每一行进行快速的分析。通常分析行开始处和结尾处的一些字节就足够，从而丢弃那些不感兴趣的行。大部分行是生成器—处理器不感兴趣的。但是如果某一行可能引起兴趣，那么就要进行更详细的处理。如果资源监听器 85 认为某一行是资源模式，那么生成器—处理器调用某个生成器—处理器特定的逻辑并且标记该资源。该生成器—处理器特定的逻辑包括往回搜索资源的开始处并且向前搜索资源的结尾处。资源被发现并且处理器通知组织器关于资源的起始位置和结束位置。组织器依照上述讨论的包装方案标记资源并且立即将其位置前进到该资源之后。这样就结束了对这个资源的处理。

[0184] 如果生成器—处理器没有识别出行，那么它就有效返回。接着，组织器使用下面描述的一般 DSC 处理器逻辑来处理行。

[0185] 这种方法的长处在于每个生成器—处理器都能在被要求时重写一般 DSC 处理器的默认行为，并且同时依靠一般 DSC 处理器的能力来处理大部分的行。在这种方式中，每个生成器—处理器可以在补偿特定的非 DSC 符合性所需的最小数量的代码行内实施；更加符合的生成器导致更加简单的生成器—处理器实施。

[0186] 继续该例子，组织器探测应用程序（驱动器 LaserWriter8 在前面的阶段中已经被探测到）。为了更有针对性，假设其为 Adobe Acrobat。组织器将这个 Adobe Acrobat 作为第二元件安装在生成器链中。由于这一点，组织器将为在生成器链中的每个生成器—处理器提供每一行：

[0187] ——组织器将为 Adobe Acrobat 提供行；

[0188] ——如果该行被拒绝，那么组织器将该行提供给 LaserWriter8；

[0189] ——如果该行被拒绝，那么组织器将使用一般（或通用）的 DSC 处理器处理该行。

[0190] 一般的 DSC 处理器

[0191] 在图 8 中，一般的 DSC 处理器 82 负责一般的 DSC 处理流程，该流程如同在“Adobe 文档结构转换规范”中所定义的一样。

[0192] 尽管不可以依赖于 DSC 符合性，如同可以在上面的“一般处理流程”描述中所见，但是一般的 DSC 处理器是非常重要的部件。该一般的 DSC 处理器实现组织器的默认行为并且将每个生成器—处理器做的尽可能小并且易于实现。一般的 DSC 处理器执行的操作例如有：分析作业头、分析序言 (prologue)、分析作业缺省、分析资源、分析指令集、寻找页面边

界、寻找作业尾和许多其他的操作，需要这些操作以用于在“Adobe 文档结构转换规范”描述的一般 DSC 处理。除此之外，一般的 DSC 处理器可以执行其他的针对实现的功能，这些功能对组织作业用于并行处理并不是严格需要的。

[0193] 创建器监听器

[0194] 创建器监听器 83 负责识别生成器链。如上所述，更加精确的是不仅仅讨论单个创建器或单个生成器而是讨论由多个生成器组成的生成链。使用％％创建器 DSC 通常是不可靠的。最可靠的方法是分析指令集 (ProcSets)，该指令集在 PostScript 作业中的特殊部分，其定义特定生成器所需的 PostScript 程序。这样，如果作业是由 LaserWriter8 驱动器生成的，那么组织器将在某个点上遇到 LaserWriter8 指令集。如果作业是由 Adobe Acrobat 应用程序生成的，那么组织器将在某个点遇到 Adobe Acrobat 指令集。在假设的例子中，如果作业由 XvzSoft 生成，但是没有 XyzSoft 指令集，那么这只是表明 XyzSoft 在本作业中不使用任何特定的 XyzSoft 资源，并且因此不需要分析 XyzSoft 模式。考虑到有多种不同的生成器，在某些情况下在确定生成器时分析％％创建器 DSC 和其他 DSC 仍然是有利的。

[0195] 页面数据监听器

[0196] 页面数据监听器 84 负责决定是否将整个页面标记为资源。显然的，该逻辑对于每个生成器而言是不同的。

[0197] 经验显示，例如，在多个 PostScript 拼版包中，对于给定的生成器通常可以实施探测和提取该生成器所用资源的部件。应当理解的是，通常来说这样做并不容易，需要投入大量的工程时间。对于 PostScript 拼版应用程序和其他寻求页面独立性的方法，没有其他的可行的选项，资源必须被探测和提取。这就是为什么应用程序通常使用下列两种方法 :1) 投入大量的努力来处理多个生成器 ;以及 2) 限制支持的生成器的数量。

[0198] 不寻求页面独立性的本发明在其处置上具有另一选项。如同在实际中显示的，识别页面上存在资源要比提取或标记这些资源明显地更容易。这就是为什么本发明的实施者可以在某些情况下选择快速通过页面，如果发现资源，那么就将整个页面标记为资源。考虑上上面的语句“资源的集中度在作业中迅速下降”，本发明的这个部分允许在非常短的时间内实施本发明非常合理的实施例。显然地，本发明更精心设计的实施例将小心地使用上述快捷方式并且将为大部分重要的生成器实施资源标记。

[0199] 资源监听器

[0200] 资源监听器 85 负责识别和标记资源。资源监听在上面已经被描述。实施者应当期望大部分时间被用在实施针对生成器的资源监听器，除非使用上述资源页面的快捷方式。考虑到实施多个拼版，本领域普通技术人员能够实施有效地实施本发明所要求的资源监听。

[0201] 图像监听器

[0202] 图像监听器 86 负责探测图像边界和有效地跳过图像。图像可能非常的大并且有效地识别和跳过它们是有利的。显然的，一般的 DSC 处理器 82 逻辑被用于根据 DSC 协定跳过图像。该逻辑需要被针对生成器的模式识别逻辑强化以适应非 DSC 符合性。

[0203] EPS 监听器

[0204] EPS 监听器 87 负责探测在 PostScript 作业内的经过封装的 PostScript(EPS) 边

界并且有效地跳过 EPS。遗憾的是，一些生成器不使用用于嵌入 EPS 片段的 DSC 机制。不能识别 EPS 和从资源解析中跳过 EPS 可能会导致不正确的解析（例如，生成多余的页面或者标记导致资源冲突的多余资源）。这就是为什么需要特别的针对生成器的模式识别逻辑来监听 EPS。

[0205] 图形状态监听器

[0206] 图形状态监听器 88 负责收集所有针对生成器的习惯用语，这些习惯用语影响持久的图形状态。需要这个针对生成器的监听器来收集所有针对生成器的习惯用语，这些习惯用语影响在页面之间持续的图形状态并且如上所述地将该图形状态与每个页面相联合。这种习惯用语的例子为由 Windows 驱动器生成的“fontname Ji”命令，“fontname Ji”是在超过页面范围外持续的 PostScript 的“setfont”（设置字体）命令的别名。

[0207] 名称列表

[0208] 11 PostScript 作业

[0209] 12 第一处理器

[0210] 13 第二处理器

[0211] 14 第三处理器

[0212] 15 第四处理器

[0213] 22 解释器

[0214] 26 呈现器

[0215] 32 中央解释器处理器

[0216] 33 独立页面

[0217] 34 呈现处理器

[0218] 42 PostScript 至 PDF 转换器

[0219] 43 PDF 作业

[0220] 44 PDF 分配器

[0221] 45 多个处理器

[0222] 51 多个 PostScript 作业

[0223] 55 多个处理器

[0224] 62 作业组织器

[0225] 63 经过组织的作业

[0226] 64 分配器

[0227] 65 多个 PDL 处理器

[0228] 66 处理器

[0229] 74 共同资源

[0230] 75 共享的资源存储器

[0231] 81 组织器

[0232] 82 一般 DSC 处理器

[0233] 83 创建器监听器

[0234] 84 页面数据监听器

[0235] 85 资源监听器

[0236] 86 图像监听器

[0237] 87 EPS 监听器

[0238] 88 图形状态监听器

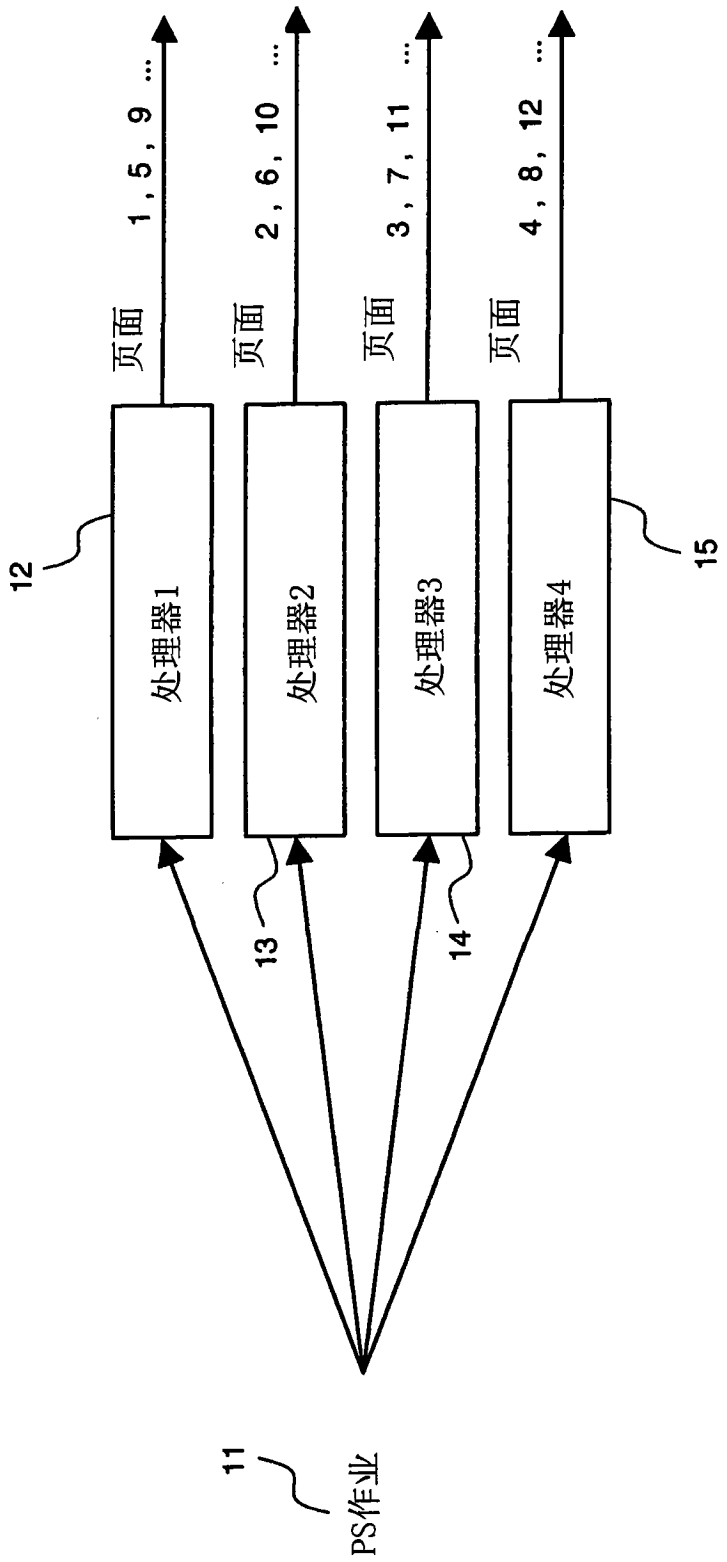

页面 1, 5, 9 ...
页面 2, 6, 10 ...
页面 3, 7, 11 ...
页面 4, 8, 12 ...
12
处理器1
处理器2
处理器3
处理器4
15
13
14
11
PS作业

图 1 （现有技术）

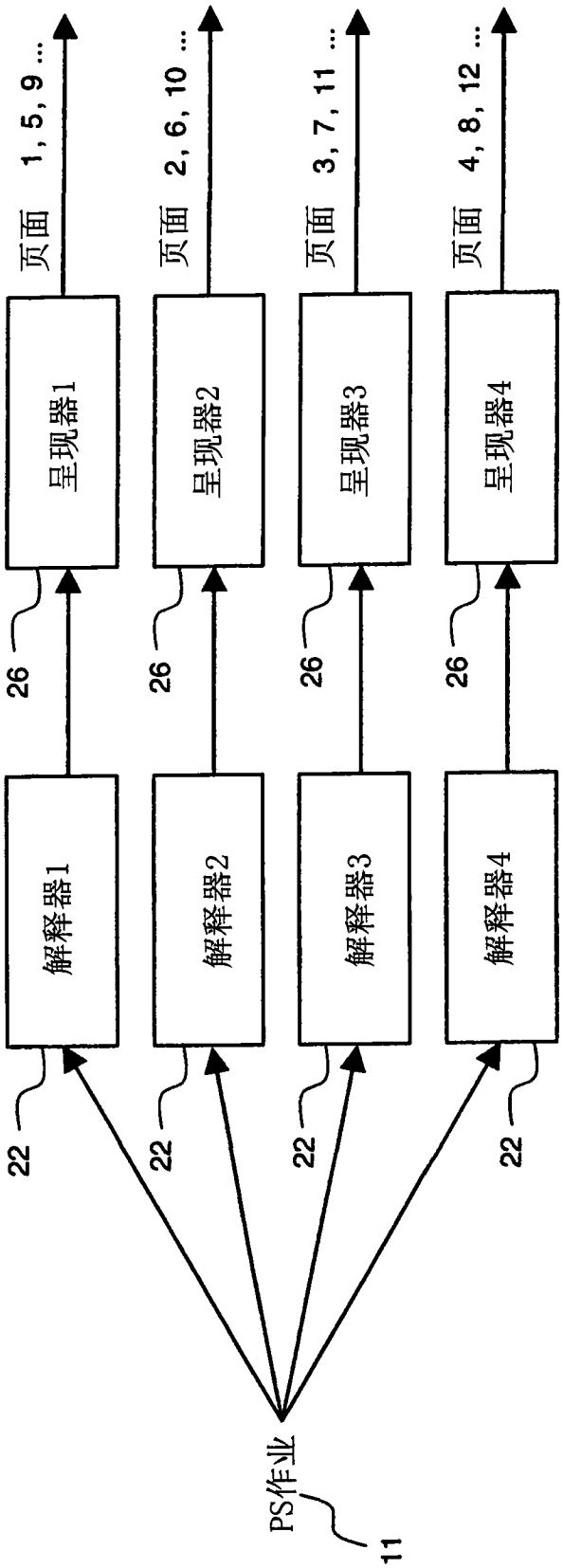

PS作业
11
22
22
22
22
解释器1
解释器2
解释器3
解释器4
26
26
26
26
呈现器1
呈现器2
呈现器3
呈现器4
页面 1, 5, 9 ...
页面 2, 6, 10 ...
页面 3, 7, 11 ...
页面 4, 8, 12 ...

图 2

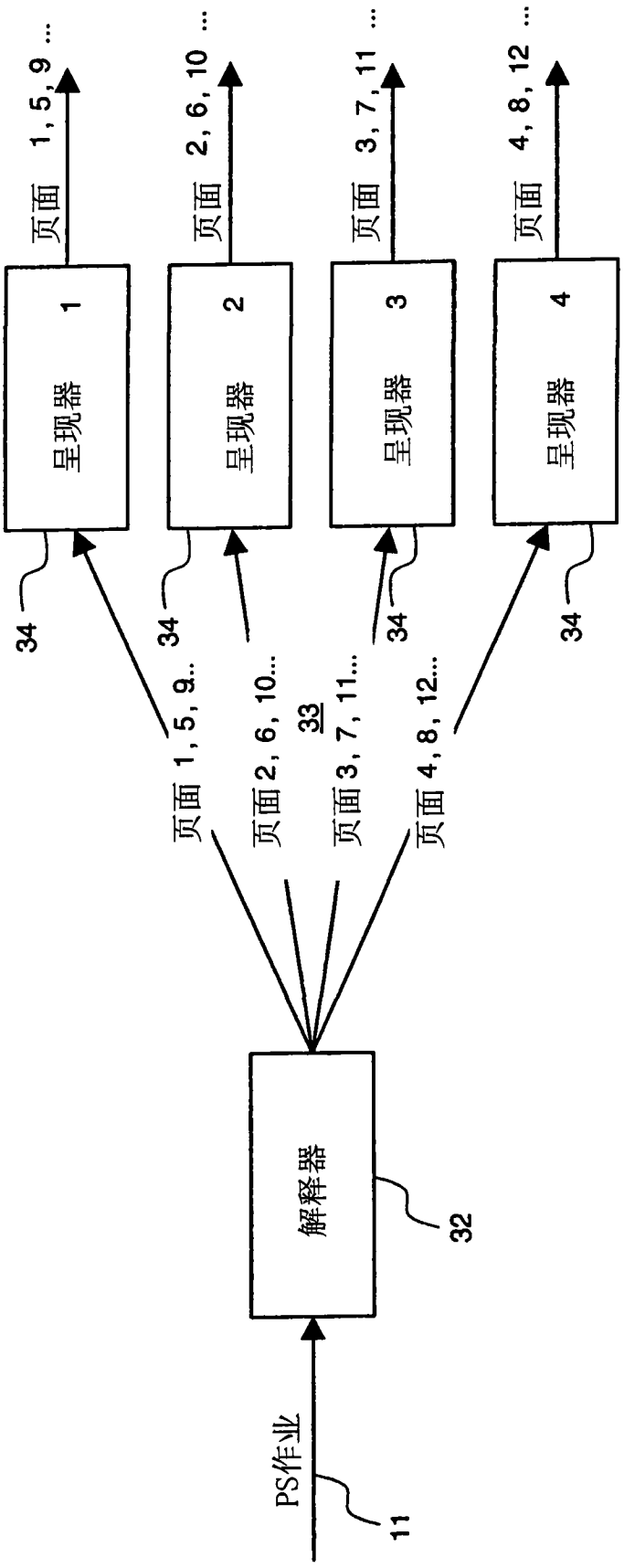

PS作业
11
解释器
32
33
呈现器 1
呈现器 2
呈现器 3
呈现器 4
页面 1, 5, 9 ...
页面 2, 6, 10 ...
页面 3, 7, 11 ...
页面 4, 8, 12 ...
34
34
34
34
页面 1, 5, 9...
页面 2, 6, 10...
页面 3, 7, 11...
页面 4, 8, 12...

图 3 （现有技术）

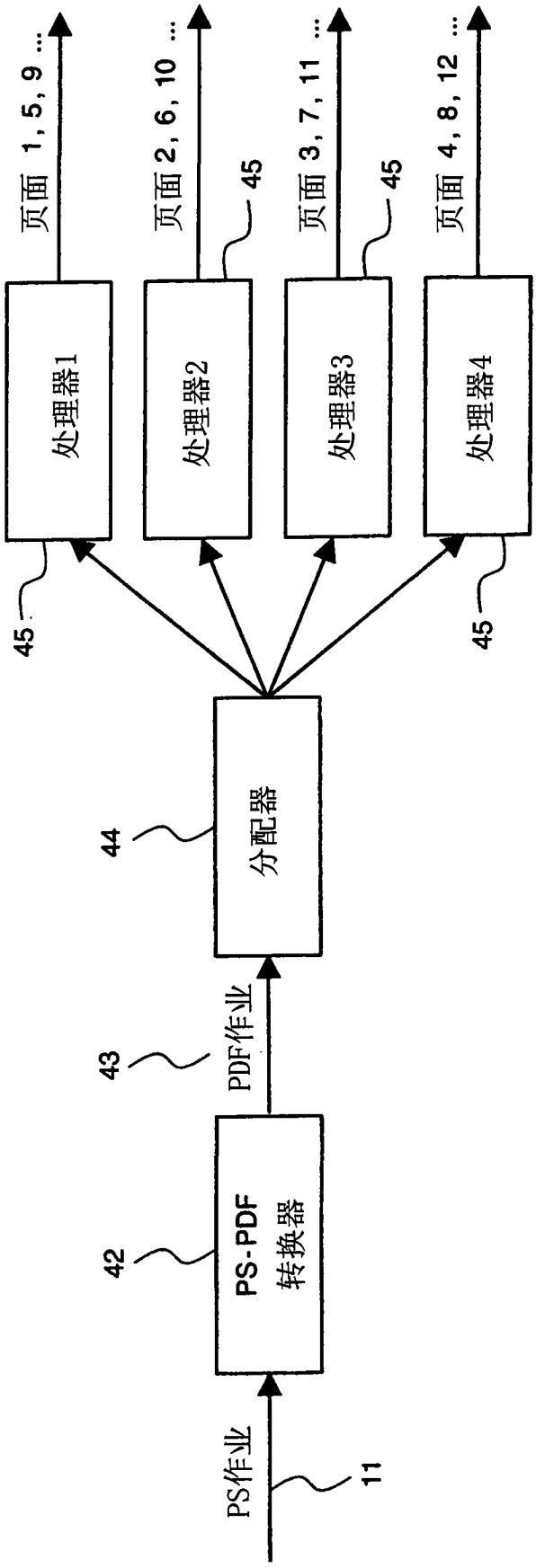

PS作业
11
42
PS-PDF
转换器
43
PDF作业
44
分配器
45
处理器1
处理器2
处理器3
处理器4
页面 1, 5, 9 ...
页面 2, 6, 10 ...
页面 3, 7, 11 ...
页面 4, 8, 12 ...

图 4

（现有技术）

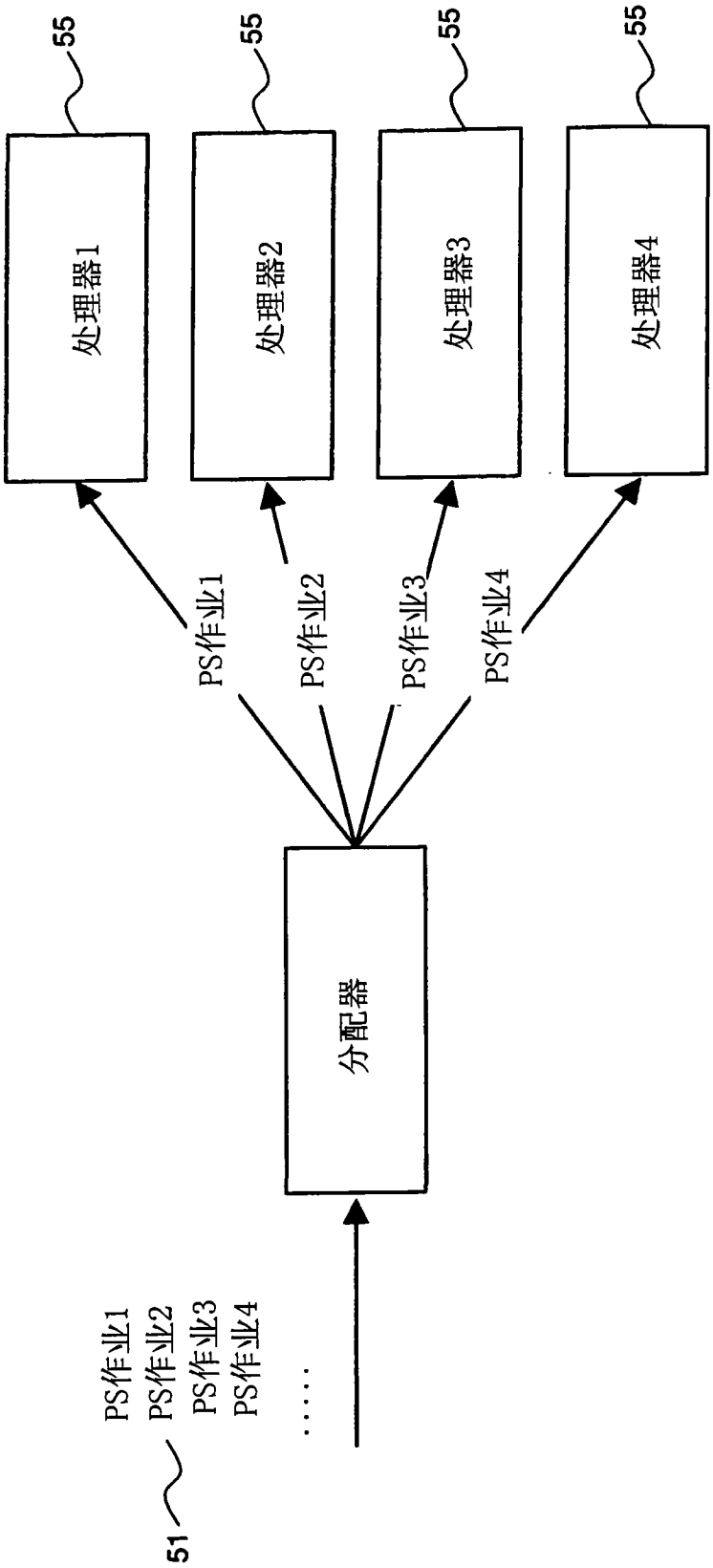

55
55
55
55
处理器1
处理器2
处理器3
处理器4
PS作业1
PS作业2
PS作业3
PS作业4
分配器
PS作业1
PS作业2
PS作业3
PS作业4
.....
51

图 5

（现有技术）

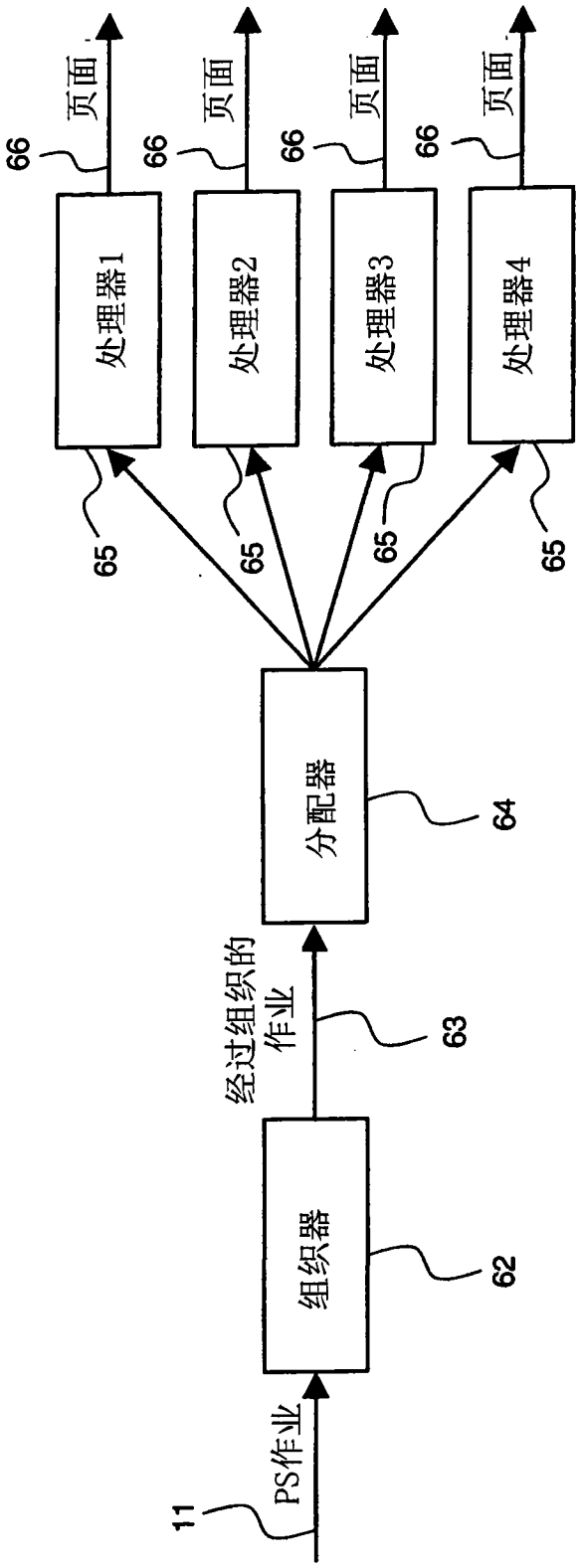


图 6

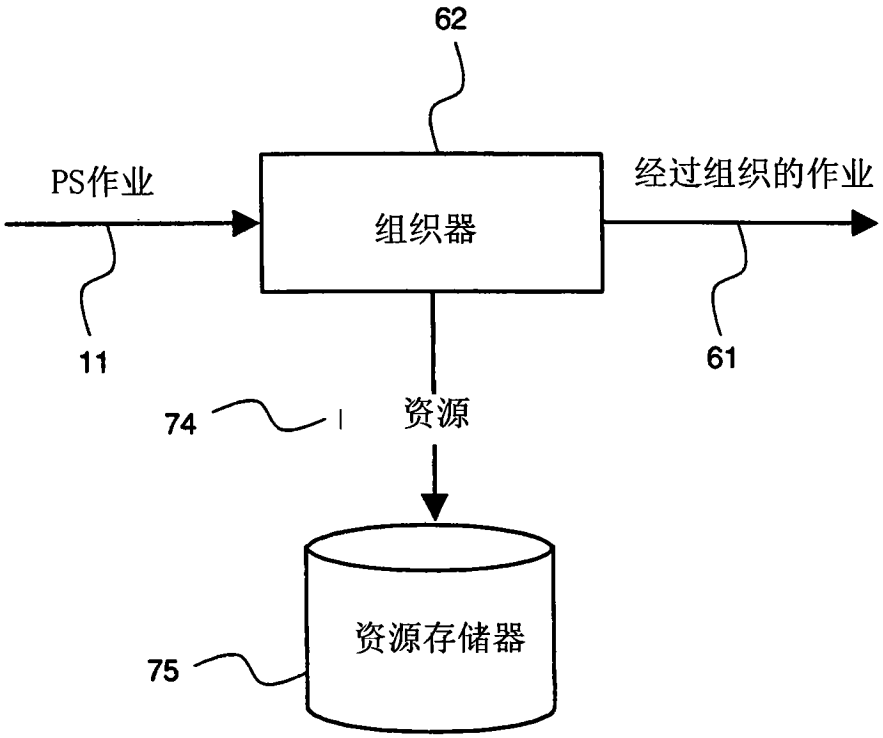


图 7

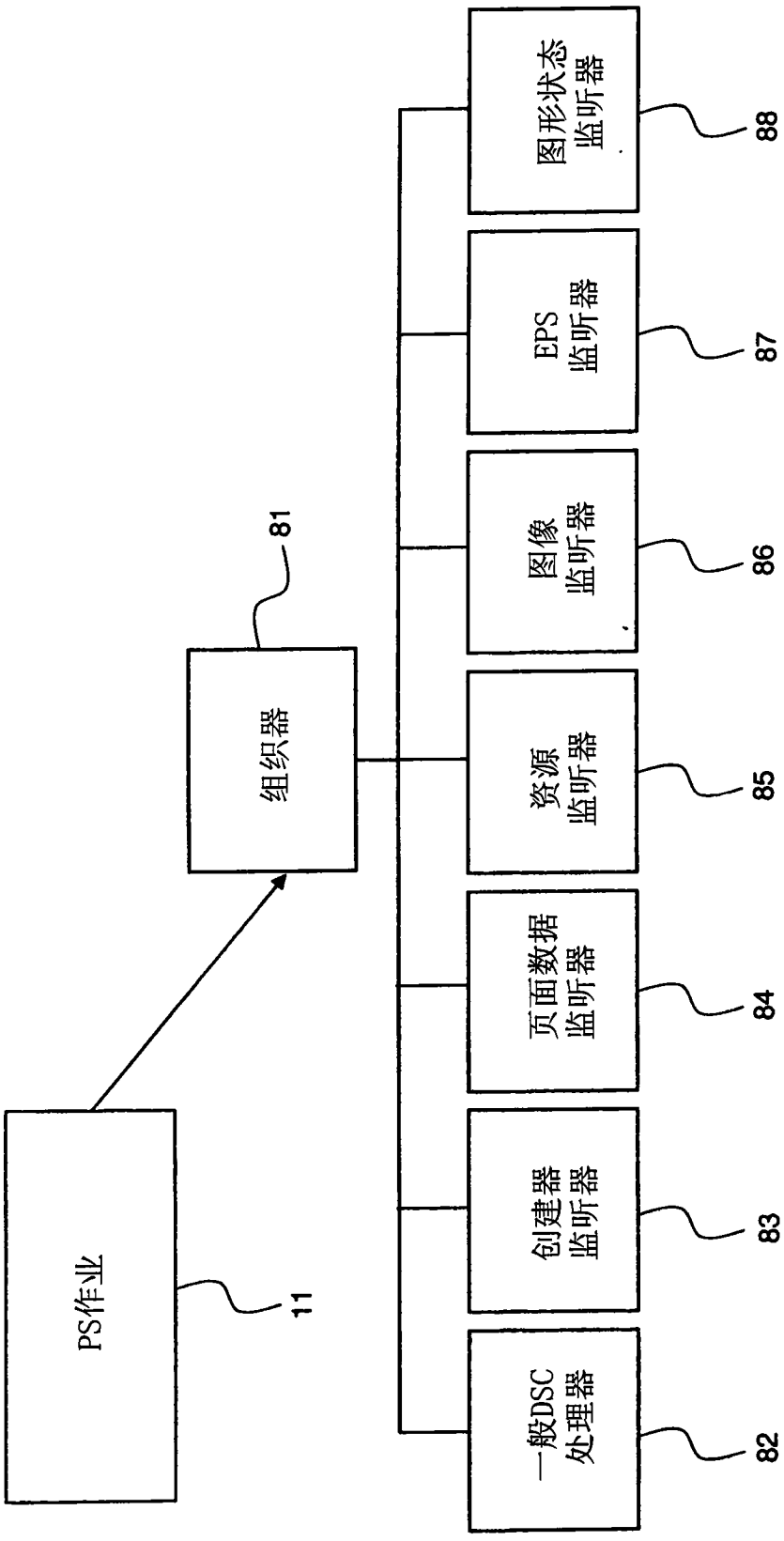

PS作业
11
组织器
81
一般DSC
处理器
82
创建器
监听器
83
页面数据
监听器
84
资源
监听器
85
图像
监听器
86
EPS
监听器
87
图形状态
监听器
88

图 8