(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 698 29 442 T2** 2006.04.13

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 889 400 B1**
(21) Deutsches Aktenzeichen: **698 29 442.4**
(96) Europäisches Aktenzeichen: **98 305 133.5**
(96) Europäischer Anmeldetag: **29.06.1998**
(97) Erstveröffentlichung durch das EPA: **07.01.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **23.03.2005**
(47) Veröffentlichungstag im Patentblatt: **13.04.2006**

(51) Int Cl.[8]: ***G06F 9/445*** (2006.01)
G06F 17/30 (2006.01)

(30) Unionspriorität:
885024 **30.06.1997** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB, NL, SE

(73) Patentinhaber:
Sun Microsystems, Inc., Santa Clara, Calif., US

(74) Vertreter:
Dr. Weber, Dipl.-Phys. Seiffert, Dr. Lieke, 65183 Wiesbaden

(72) Erfinder:
Viswanathan, Srinivasan, Fremont, California 94536, US; Nazari, Siamak, Arcadia, California 91006, US; Swaroop, Anil, Loma Linda, California 92354, US; Khalidi, Yousef, Sunnyvale, California 94086, US

(54) Bezeichnung: **System und Verfahren für transparenten, globalen Zugang zu physikalischen Geräten in einem Rechnersystem**

Beschreibung

[0001] Die vorliegende Erfindung bezieht sich allgemein auf Systeme und Verfahren zum Zugriff auf physikalische Geräte bzw. Einrichtungen, die an einen Computer angeschlossen sind und insbesondere auf Systeme und Verfahren zum Zugriff auf physikalische Einrichtungen an einem Computer-Cluster.

HINTERGRUND DER ERFINDUNG

[0002] Es ist für Unix-basierte Computeranwendungen immer üblicher geworden, daß sie auf einem Cluster beheimatet sind, der eine Mehrzahl von Computern umfaßt. Es ist das Ziel von Cluster-Betriebssystemen, den Betrieb des Clusters für Anwendungen/Benutzer so transparent zu machen, als wenn es sich um einen einzelnen Computer handeln würde. Zum Beispiel stellt ein Cluster typischerweise ein globales Dateisystem bereit, das einen Benutzer in die Lage versetzt, alle herkömmlichen Dateien auf dem Cluster zu sehen und auf sie zuzugreifen unabhängig davon, wo die Dateien gelegen sind. Diese Transparenz gilt jedoch nicht für einen Gerätezugriff auf einem Cluster.

[0003] Typischerweise wird Gerätezugriff auf Unix-basierten Systemen durch eine spezielles Dateisystem (z.B. SpecFS) bereitgestellt, das Geräte wie Dateien behandelt. Dieses spezielle Dateisystem wird nur auf einem einzelnen Knoten betrieben. Das bedeutet, daß es nur einen Benutzer auf einem bestimmten Knoten in die Lage versetzt, Geräte an diesem Knoten zu sehen und auf sie zuzugreifen, was dem Ziel einer globalen Sichtbarkeit von Geräten auf einem Cluster zuwiderläuft. Diese Einschränkungen rühren sowohl von der fehlenden Koordination zwischen den speziellen Dateisystemen, die auf unterschiedlichen Knoten laufen, als auch von der fehlenden Strategie für das Benennen von Geräten her, um der globalen Sichtbarkeit von Geräten bzw. Einrichtungen Rechnung zu tragen. Diese Aspekte eines Zugriffsystems auf Geräte nach dem Stand der Technik werden nun unter Bezug auf die Fig. 1–Fig. 4 beschrieben.

[0004] In Fig. 1 ist ein Blockdiagramm eines herkömmlichen Computersystems **100** abgebildet, das eine zentrale Verarbeitungseinheit (Central Processing Unit, CPU) **102**, einen Hochgeschwindigkeitsspeicher **104**, eine Mehrzahl von physikalischen Einrichtungen **106** und eine Gruppe von physikalischen Geräteschnittstellen **108** beinhaltet (z. B. Busse oder andere elektronische Schnittstellen), die die CPU **102** in die Lage versetzen, Daten zu steuern und mit dem Speicher **102** und den physikalischen Einrichtungen **106** auszutauschen. Der Speicher **102** kann ein wahlfrei zugreifbarer Speicher (Random Access Memory, RAM) oder ein Cachespeicher sein.

[0005] Die physikalischen Einrichtungen **106** können Hochverfügbarkeitseinrichtungen **112**, Drucker **114**, Kernspeicher **116**, Kommunikationseinrichtungen **118** und Speichereinrichtungen **120** (z. B. Plattenlaufwerke) umfassen, sind jedoch nicht darauf beschränkt. Drucker **114** und Speichereinrichtungen **120** sind wohlbekannt. Hochverfügbarkeitseinrichtungen **112** umfassen Einrichtungen wie Speichereinheiten oder Drucker, die über zugeordnete Sekundär- bzw. Ersatzeinrichtungen verfügen. Solche Einrichtungen sind hochverfügbar, da die Sekundäreinrichtungen beim Ausfall einer Primäreinrichtung für ihre entsprechenden Primär- bzw. Haupteinrichtungen einspringen können. Der Kernspeicher **116** ist ein programmierter Bereich des Speichers **102**, der das Erfassen und Berichten von Statistiken zur Systemleistung bzw. zum Systemdurchsatz beinhaltet. Die Kommunikationseinrichtungen **118** umfassen Modems, ISDN-Schnittstellenkarten, Netzschnittstellenkarten und andere Arten von Kommunikationseinrichtungen. Die Einrichtungen bzw. "Geräte" **106** können auch Pseudo-Einrichtungen **122** umfassen, die Software-Einrichtungen sind, die nicht einer tatsächlichen physikalischen Einrichtung bzw. einem tatsächlichen physikalischen Gerät zugeordnet sind.

[0006] Der Speicher **104** des Computers **100** kann ein Betriebssystem **130**, Anwendungsprogramme **150** und Datenstrukturen **160** speichern. Das Betriebssystem **130** wird in der CPU **102** ausgeführt, solange der Computer **100** in Betrieb ist und Systemdienste für den Prozessor **102** und die in der CPU **102** ausgeführten Anwendungen **150** bereitstellt. Das Betriebssystem **130**, das auf V. 2.6 des auf Sun®-Workstations eingesetzten Solaris™-Betriebssystems modelliert ist, beinhaltet einen Kern **132**, ein Dateisystem **134**, Gerätetreiber **140** und einen Gerätetreiberschnittstellen-(device driver interface, DDI)-Rahmen **142**. Solaris und Sun sind Marken bzw. eingetragene Marken von Sun Microsystems Inc. Der Kern **116** behandelt Systemaufrufe von den Anwendungen **150** wie etwa Anforderungen, auf den Speicher **104**, das Dateisystem **134** oder die Einrichtungen **106** zuzugreifen. Das Dateisystem **134** und seine Beziehung zu den Geräten **106** und den Gerätetreibern **140** wird unter Bezug auf die Fig. 2A und Fig. 2B beschrieben.

[0007] In Fig. 2A ist eine Darstellung des Dateisystems **134** auf höherer Ebene abgebildet, das von V. 2.6 und früheren Versionen des Solaris-Betriebssystems eingesetzt wird. In Solaris ist das Dateisystems **134** das Medium, durch das auf alle Dateien, Geräte **106** und Netzschnittstellen (unter der Annahme, daß der Computer

100 an ein Netz angeschlossen ist) zugegriffen wird. Diese drei unterschiedlichen Arten von Zugriffen werden entsprechend von drei Komponenten des Dateisystems **134** bereitgestellt: einem Unix-Dateisystem **138u** (UFS), einem speziellen Dateisystem **138s** (SpecFS) und einem Netz-Dateisystem **138n** (NFS).

[0008] In Solaris greift eine Anwendung **150** anfänglich auf eine Datei, ein Gerät oder eine Netzschnittstelle (alle hier als ein Ziel bezeichnet) zu, indem sie eine Öffnen- bzw. Open-Anforderung für das Ziel an das Dateisystem **134** über den Kern **132** absetzt. Das Dateisystem **134** leitet dann die Anforderung je nachdem an UFS **138u**, SpecFS **138s** oder NFS **138n** weiter. Wenn das Ziel erfolgreich geöffnet ist, gibt das UFS, SpecFS oder NFS ein vnode-Objekt **136** zurück, das auf die bzw. den angeforderte(n) Datei, Einrichtung oder Netzknoten abgebildet wird. Das Dateisystem **134** bildet dann das vnode-Objekt **136** auf einen Datei-Deskriptor **174** ab, der an die Anwendung **150** über den Kern **132** zurückgegeben wird. Die anfordernde Anwendung verwendet anschließend den Datei-Deskriptor **174**, um auf die bzw. den entsprechende(n) Datei, Einrichtung oder Netzknoten, die bzw. der mit dem zurückgelieferten vnode-Objekt **136** verbunden ist, zuzugreifen.

[0009] Das vnode-Objekt **136** stellt einen generischen Satz von Dateisystem-Diensten gemäß einer vnode/VFS-Schnittstelle oder -Schicht (VFS) **172** zur Verfügung, die als die Schnittstelle zwischen dem Kern **132** und dem Dateisystem **134** dient. Solaris stellt ferner inode-, snode- und mode-Objekte **136i**, **136s**, **136r** bereit, die von dem vnode-Objekt **136** erben und auch Methoden und Datenstrukturen beinhalten, die für die Typen von Zielen angepaßt sind, die dem UFS, SpecFS bzw. NFS zugeordnet sind. Diese Klassen **136i**, **136s** und **136** bilden die Schnittstellen auf unterer Ebene zwischen den vnodes **136** und ihren jeweiligen Zielen. Somit ist, wenn das UFS, SpecFS oder NFS ein vnode-Objekt zurückliefert, dieses Objekt einem entsprechenden inode, snode oder mode zugeordnet, der die tatsächlichen Zieloperationen durchführt. Nachdem die allgemeine Struktur des Solaris-Dateisystems diskutiert wurde, richtet sich der Fokus der vorliegenden Diskussion nun auf die von Solaris verwendeten Datei-basierten Gerätezugriffsmethoden.

[0010] In Fig. 2B geben Solaris-Anwendungen **150** typischerweise Anforderungen für Gerätezugriffe an das Dateisystem **134** aus (über den Kern **132**), indem sie den logischen Namen **166** des Gerätes verwenden, das sie geöffnet benötigen. Zum Beispiel könnte eine Anwendung **150** Zugriff auf eine SCSI-Einrichtung anfordern mit dem Befehl: open (/dev/dsk/disk_logical_address).

[0011] Der logische Name /dev/dsk/disk_logical_address zeigt an, daß die zu öffnende Einrichtung eine Platte an einer bestimmten logischen Adresse ist. In Solaris könnte die logische Adresse für eine SCSI-Platte "c0t0d0sx" lauten, wobei "c0" die SCSI-Steuerung 0, "t0" das Ziel 0, "d0" die Platte 0 und "sx" die x-te Scheibe bzw. Slice für die bestimmte Platte repräsentieren (ein SCSI-Plattenlaufwerk kann bis zu acht Slices haben).

[0012] Der logische Name wird von einem der Linkgeneratoren **144** zugewiesen, die im Benutzerraum gelegene Erweiterungen des DDI-Rahmens **142** sind, und basiert auf Information, die von dem Gerätetreiber **140** beim Hinzufügen des Gerätes bzw. der Einrichtung übergeben wird, und auf einem entsprechenden physikalischen Namen für das Gerät, der von dem DDI-Rahmenwerk **142** erzeugt wird. Wenn eine Instanz eines bestimmten Gerätetreibers **140** einem Knoten **100** zugeordnet ist, ruft das DDI-System **142** die Hinzufügen- bzw. Attach-Routine dieses Treibers **140** auf. Der Treiber **140** weist dann einen eindeutigen lokalen Bezeichnen zu und ruft die ddi-create-minor-nodes-Methode **146** des DDI-Rahmens **142** für jedes Gerät auf, das mit dieser Instanz in Verbindung gebracht werden kann. Typischerweise stellt der eindeutige lokale Bezeichner einen untergeordneten Namen bzw. Minor-Namen (Zusatzbezeichnung bzw. Nebennamen) (z. B. "a") und eine untergeordnete Nummer bzw. Minor-Nummer (Zusatzziffer bzw. Nebennummer) (z. B. "2") dar. Jedesmal, wenn sie aufgerufen wird, erzeugt die ddi-create-minor-nodes-Methode **146** einen Blattknoten in dem DevInfo-Baum **162**, der ein gegebenes Gerät repräsentiert. Weil zum Beispiel ein SCSI-Laufwerk (d. h. eine Instanz) bis zu acht Slices bzw. Teile (d. h. Geräte) haben kann, weist der lokale SCSI-Treiber **140** eindeutige lokale Bezeichner jedem der acht Slices zu und ruft die ddi-create-minor-nodes-Methode **146** mit den lokalen Bezeichnern bis zu acht Mal auf.

[0013] Ebenso ist jedem Gerät **106** eine UFS-Datei **170** zugeordnet, die Konfigurierungsinformation für das Zielgerät **106** bereitstellt. Der Name einer bestimmten UFS-Datei **170i** ist derselbe wie ein physikalischer Name **168i**, der vom physikalischen Ort des Gerätes auf dem Computer abgeleitet ist. Zum Beispiel könnte ein SCSI-Gerät den folgenden physikalischen Namen **168**, /devices/iommu/sbus/esp1/sd@addr:minor_name, haben, wobei addr die Adresse des Gerätetreibers sd und minor_name der Minor-Name der Geräteinstanz ist, der von dem Gerätetreiber sd zugewiesen ist. Wie physikalische Namen abgeleitet werden, wird unten unter Bezug auf Fig. 3 beschrieben.

[0014] Um das Dateisystem **134** in die Lage zu versetzen, ein Zielgerät unter Angabe des logischen Namens

des Zielgerätes zu öffnen, verwendet es eine Datenstruktur **164** des logischen Namensraumes, die logische Dateinamen **166** auf physikalische Dateinamen **168** abbildet. Die physikalischen Namen von Geräten **106** sind von dem Ort des Gerätes in einem Geräteinformationsbaum (DevInfo tree) **140** (in Fig. 1 abgebildet) abgeleitet, der die Hierarchie von Gerätetypen, Busverbindungen, Steuerungen, Treibern und dem Computersystem **100** zugeordneten Geräten repräsentiert. Jede durch einen physikalischen Namen **168** identifizierte Datei **170** enthält in ihren Attributen einen Bezeichner oder dev_t (kurz für Device Type bzw. Gerätetyp), der dem Zielgerät eindeutig zugeordnet ist. Dieser dev_t-Wert wird von dem Dateisystem **134** verwendet, um über das SpecFS **138s** auf das richtige Zielgerät zuzugreifen. Es wird nun unter Bezug auf Fig. 3 beschrieben, wie dev_t-Werte zugewiesen werden und der DevInfo-Baum **140** von dem DDI-System **142** auf dem Laufenden gehalten wird.

[0015] In Fig. 3 wird eine Darstellung eines hypothetischen DevInfo-Baumes **162** für das Computersystem **100** gezeigt. Jeder Knoten des DevInfo-Baumes **162** entspricht einer physikalischen Komponente des Gerätesystems, die dem Computer **100** zugeordnet ist. Unterschiedliche Ebenen bzw. Niveaus entsprechen unterschiedlichen Ebenen der Gerätehierarchie. Knoten, die direkt mit einem höheren Knoten verbunden sind, stellen Objekte dar, die Instanzen des Objektes auf höherer Ebene sind. Folglich ist der Wurzelknoten des DevInfo-Baumes immer der "/"-Knoten, unter dem die gesamte Gerätehierarchie angesiedelt ist. Die Zwischenknoten (d. h. Knoten, die keine Blattknoten und Blatt-Eltern-Knoten sind) werden als Nexus- bzw. Verknüpfungsgeräte bezeichnet und entsprechen dazwischenliegenden Strukturen wie Steuerungen, Bussen und Anschlüssen bzw. Ports. Auf der nächsten Ebene oberhalb der untersten des DevInfo-Baumes befinden sich die Gerätetreiber, von denen jeder ein oder mehrere Geräte exportieren oder kontrollieren kann. Auf der Blattebene sind die tatsächlichen Geräte bzw. Einrichtungen, von denen jedes abhängig vom Gerätetyp eine Anzahl von Geräteinstanzen exportieren kann. Zum Beispiel kann eine SCSI-Einrichtung bis zu sieben Instanzen haben.

[0016] Der in Fig. 3 gezeigte hypothetische DevInfo-Baum **162** repräsentiert ein Computersystem **100**, das eine Eingabe-/Ausgabe-(I/O)-Steuerung für auf Speicher abgebildete I/O-Einrichtungen (iommu) an einer physikalischen Adresse addr0 beinhaltet. Die iommu steuert bzw. verwaltet die Interaktionen der CPU mit I/O-Einrichtungen, die mit einem Systembus (sbus) an Adresse addr1 und einem Hochgeschwindigkeitsbus wie einem PCI-Bus an Adresse addr2 verbunden sind. Zwei SCSI-Steuerungen (esp1 und esp2) an entsprechenden Adressen addr3 und addr4 sind zusammen mit einer Steuerung für einen Asynchron-Übertragungsmodus (Asynchronous Transfer Mode, ATM) an addr5 mit dem sbus verbunden. Die erste SCSI-Steuerung esp1 ist einem SCSI-Gerätetreiber (sd) an Adresse 0 (dargestellt als @0) zugeordnet, der vier SCSI-Geräteinstanzen (dev0, dev1, dev2, dev3) verwaltet. Jede dieser Geräteinstanzen einspricht einer entsprechenden Slice bzw. Scheibe eines einzelnen, physikalischen Gerätes **106**. Die erste SCSI-Steuerung esp1 ist auch einem SCSI-Gerätetreiber (sd) an Adresse 1 zugeordnet, der mehrere SCSI-Geräteinstanzen (nicht abgebildet) eines anderen physikalischen Gerätes **106** steuert.

[0017] Jedem Typ von Gerätetreiber, der bei dem Computersystem **100** verwendet werden kann, ist eine vorher festgelegte, eindeutige übergeordnete Nummer bzw. Major-Nummer (Hauptnummer) zugewiesen. Zum Beispiel ist dem SCSI-Gerätetreiber sd die Major-Nummer **32** zugewiesen. Jedes Gerät ist einer Minor-Nummer zugeordnet, die innerhalb der Gruppe von Geräten, die von einem einzelnen Gerätetreiber gesteuert werden, eindeutig ist. Zum Beispiel haben die dem Treiber sd an Adresse 0 zugeordneten Geräte dev0, dev1, dev2 und dev3 die Minor-Nummern 0, 1, 2 bzw. 3 und Minor-Namen a, b, c bzw. d. In ähnlicher Weise würden die von dem Treiber sd an Adresse 1 gesteuerten Geräte Minor-Nummern haben, die von den den Geräten dev0–dev3 zugeordneten verschieden sind (z. B. vier davon könnten Minor-Nummern **4-7** haben). Die Minor-Nummern und -Bezeichnungen werden von dem Eltern-Gerätetreiber **140** (Fig. 1) für jede neue Geräteinstanz zugewiesen (es sei daran erinnert, daß eine SCSI-Instanz ein spezielles SCSI-Laufwerk und ein SCSI Gerät eine bestimmte Slice dieses Gerätes sein könnten). Dies stellt sicher, daß jedes von einem gegebenen Treiber exportierte Gerät eine eindeutige Minor-Nummer und -Bezeichnung hat. Das bedeutet, ein Treiber verwaltet einen Minor-Nummern-Bezeichnungs-Raum.

[0018] Jede Minor-Nummer bildet, wenn sie mit der Major-Nummer seines Elterntreibers kombiniert wird, einen dev_t-Wert, der jedes Gerät eindeutig identifiziert. Zum Beispiel haben die von dem Treiber sb an Adresse 0 gesteuerten Geräte dev0, dev1, dev2 und dev3 entsprechende dev_t-Werte von (32,0), (32,1), (32,2) und (32,3). Das SpecFS **138s** unterhält eine Abbildung von dev_t-Werten zu den entsprechenden Geräten bzw. Einrichtungen. Im Ergebnis bezeichnen alle Anforderungen an das SpecFS zum Öffnen eines Gerätes das zu öffnende Gerät mittels seines eindeutigen dev_t-Wertes.

[0019] Der DevTree-Pfad zu einem Gerät liefert den physikalischen Namen dieses Gerätes. Zum Beispiel ist der physikalische Name des Gerätes dev0 gegeben durch die Zeichenkette: /devices/iom-

mu@addr0/sbus@addr1/esp1@addr3/sd@0:a, wobei sich sd@0:a auf das Gerät bezieht, das von dem sd-Treiber an Adresse 0 gesteuert wird, dessen Minor-Name a ist; d. h. das Gerät dev0. Der physikalische Name bezeichnet die spezielle Datei **170** (in **Fig.** 2 abgebildet) (einem snode entsprechend), die alle Information enthält, die zum Zugriff auf das entsprechende Gerät notwendig ist. Unter anderem enthalten die Attribute jeder speziellen Datei **170** den dev_t-Wert, der dem entsprechenden Gerät zugeordnet ist.

[0020] Wie oben erwähnt, erzeugt ein link_generator **144** den logischen Namen eines Gerätes aus dem physikalischen Namen des Gerätes gemäß einem Satz von Regeln, die auf von diesem Verknüpfungs- bzw. Link-Generator verwaltete Geräte anwendbar sind. Zum Beispiel könnte in dem Fall des Gerätes dev0, das von dem Treiber sd an Adresse 0 verwaltet wird, ein Link-Generator für SCSI-Geräte den folgenden logischen Namen erzeugen, /dev/dsk/c0t0d0s0, wobei sich c0 auf die Steuerung esp1@addr3 bezieht, t0 auf die Ziel-Id der physikalischen Platte, die von dem sd@0-Treiber gesteuert wird, d0 auf den sd@0-Treiber und s0 den Abschnitt bzw. die Slice mit der Minor-Bezeichnung a und der Minor-Nummer 0 bezeichnet. Das dem sd@1-Treiber zugeordnete Gerät dev0 könnte den logischen Namen /dev/dsk/c0t1d1s4 von demselben Link-Generator **144** zugewiesen bekommen. Man beachte, daß die beiden dev0-Geräte logische Namen haben, die sich in Differenzen in den Ziel-, Platten- und Slicewerten unterscheiden. Es wird nun unter Bezug auf **Fig. 4** beschrieben, wie diese Infrastruktur gegenwärtig in Solaris verwendet wird, um eine Anwendung in die Lage zu versetzen, ein bestimmtes, auf einem Computer **100** befindliches Gerät zu öffnen.

[0021] In **Fig. 4** ist ein Flußdiagramm von Operationen abgebildet, die im Speicher **104** des Computers **100** von verschiedenen Betriebssystemkomponenten während des Öffnens eines Gerätes, wie von einer Anwendung **150** angefordert, durchgeführt werden. Der Speicher **104** ist in einen Nutzerspeicherbereich **104U**, in dem die Anwendungen **150** ausgeführt werden, und in einen Kernspeicherbereich **104K** unterteilt, in dem die Betriebssystemkomponenten ausgeführt werden. Dieses Diagramm zeigt mit einem Satz von beschrifteten Pfeilen die Reihenfolge, in der die Operationen auftreten, und die Einrichtungen, die Ursprung oder Ziel jeder Operation sind. Wo es zutrifft, geben gestrichelte Linien ein Objekt an, auf das eine Referenz übergeben wird. Neben der Darstellung des Speichers **104** ist jede Operation, die einem beschrifteten Pfeil zugeordnet ist, definiert. Die Operationen sind als Nachrichten oder Funktionsaufrufe definiert, wobei auf den Nachrichtennamen die Daten folgen, die von der empfangenden Einheit verarbeitet oder zurückgegeben werden. Die Nachricht (**4-1**), "open(logical_name)", ist zum Beispiel die von der Anwendung **150** ausgegebene Nachricht, die den Kern **132** auffordert, das in dem Nutzerspeicherbereich **104U** durch "logical_name" repräsentierte Gerät zu öffnen. In diesem speziellen Beispiel ist die Anwendung bestrebt, das Gerät dev2 zu öffnen.

[0022] Nach Empfang der Öffnen-Nachricht (**4-1**) setzt der Kern **132** die Nachricht (**4-2**), "get_vnode(logival_name)", an das Dateisystem **134** ab. Diese Nachricht fordert das Dateisystem **134** auf, den vnode des Gerätes dev2 zurückzugeben, den der Kern **132** benötigt, um die Öffnen-Operation abzuschließen. Als Reaktion darauf konvertiert das Dateisystem **134** den logischen Namen **166** in den entsprechenden physikalischen Namen **168** mittels des logischen Namensraumes **164**. Das Dateisystem **134** lokalisiert dann die Datei, die von dem physikalischen Namen bezeichnet wird, und bestimmt den dev_t-Wert des zugehörigen Gerätes aus den Attributen dieser Datei. Sobald es den dev_t-Wert erhalten hat, setzt das Dateisystem **134** die Nachricht (**4-3**), "get_vnode(dev_t)", an das SpecFS **138s** ab. Diese Nachricht fordert das SpecFS **138s** auf, eine Referenz auf den mit dem Gerät dev2 verbundenen vnode zurückzuliefern. Auf den Empfang der Nachricht (**4-3**) hin erzeugt das SpecFS **138s** den angeforderten vnode **136** und einen snode **136s**, der den vnode **136** an Gerät dev2 bindet, und liefert die Referenz auf den vnode **136** (**4-4**) an das Dateisystem **134** zurück. Das Dateisystem **134** gibt dann die vnode-Referenz an den Kern zurück (**4-5**).

[0023] Sobald er die vnode-Referenz hat, setzt der Kern **132** eine Anforderung (**4-6**) an das SpecFS **138s** ab, das dem vnode **136** zugeordnete Gerät dev2 zu öffnen. Das SpecFS **138s** versucht, diese Anforderung durch Absetzen eines Öffnen-Kommandos (**4-7**) an den Treiber **2**, von dem das SpecFS weiß, daß er das Gerät dev2 steuert, zu erfüllen. Wenn der Treiber **2** in der Lage ist, das Gerät dev2 zu öffnen, gibt er eine open_status-Nachricht (**4-8**) zurück, die anzeigt, daß die Öffnen-Operation erfolgreich war. Ansonsten gibt der Treiber **2** eine Fehleranzeige in derselben Nachricht (**4-8**) zurück. Das SpecFS **138s** gibt dann eine ähnliche Statusmeldung (**4-9**) direkt an den Kern **132** zurück. Unter der Annahme, daß "Erfolg" in der Nachricht (**4-9**) zurückgegeben wurde, gibt der Kern **132** einen Datei-Deskriptor an die Anwendung **150** zurück, der eine Repräsentation des mit dem Gerät dev2 verbundenen vnode **136** im Nutzerspeicherraum ist (**4-10**). Die Anwendung **150** kann, sobald sie im Besitz des Datei-Deskriptors ist, auf das Gerät dev2 über den Kern **132** und das Dateisystem **134** mittels Dateisystemoperationen zugreifen. Die Anwendung **150** bearbeitet zum Beispiel Eingabedaten von dem Gerät dev2, indem sie an den zurückgelieferten Datei-Deskriptor gerichtete Leseanforderungen absetzt. Diese Dateisystem-Kommandos werden dann von dem SpecFS **138s** in tatsächliche Gerätekommandos und die vnode- und snode-Objekte **136**, **136s** umgewandelt, die das Gerät dev2 verwalten.

[0024] Folglich setzt Solaris Benutzer eines Computersystems **100** in die Lage, auf Einrichtungen an diesem System **100** relativ einfach zuzugreifen. Jedoch erlauben es die von Solaris eingesetzten Methoden Benutzern nicht, auf Einrichtungen transparent über Computer hinweg zuzugreifen, auch wenn die verschiedenen Computer als Teil eines Cluster konfiguriert sind. Das bedeutet, daß eine Anwendung, die auf einem ersten Computer läuft, mittels Solaris ein Gerät auf einem zweiten Computer nicht transparent öffnen kann.

[0025] Der Grund dafür, daß die aktuelle Version von Solaris keinen transparenten Gerätezugriff in der Mehr-Computer-Situation zur Verfügung stellen kann, hat mit der Art und Weise zu tun, in der die dev_t- und Minor-Nummern derzeit zugewiesen werden, wenn Geräte hinzugefügt werden. Gemäß **Fig. 3** weist jedesmal, wenn ein Gerät zu einem Computer **100** hinzugefügt wird, der zugeordnete Treiber des Gerätes diesem Gerät eine Minor-Nummer zu, die innerhalb der Menge von Geräten, die von diesem Treiber gesteuert werden, eindeutig ist und daher auf einen eindeutigen dev_t-Wert für den Computer **100** abgebildet werden können, wenn sie mit der Major-Nummer des Treibers kombiniert wird. Wenn jedoch dieselben Geräte und Treiber auf einem zweiten Computer bereitgestellt würden, würden dem Treiber und den Geräten ein ähnlicher, wenn nicht identischer Satz von Major- und Minor-Nummern und dev_t-Werten zugewiesen. Wenn zum Beispiel beide Computer einen SCSI-Treiber sd (Major-Nummer = 32) und vier SCSI-Geräteinstanzen hätten, die von dem SCSI-Treiber sd verwaltet würden, würde jeder Treiber sd denselben Satz von Minor-Nummern ihrem lokalen Satz von SCSI-Geräten zuordnen (z. B. würden beide Sätze Minor-Nummern zwischen 0 und 3 haben). Folglich würde unter Beachtung, daß auf ein Gerät anhand seines dev_t-Wertes zugegriffen wird, wenn eine Anwendung auf einem ersten Knoten eine SCSI-Platte auf dem zweiten Knoten öffnen möchte, diese Anwendung nicht in der Lage sein, die SCSI-Platte gegenüber dem SpecFS auf beiden Computersystemen eindeutig zu bezeichnen.

[0026] Daher besteht ein Bedarf für ein Datei-basiertes Gerätezugriffssystem, das Anwendungen in die Lage versetzt, wo immer sie auch ausgeführt werden, transparent auf Geräte zuzugreifen, die sich auf irgendeinem Knoten eines Computer-Clusters befinden.

[0027] EP-A-0780778 offenbart ein System und ein Verfahren zum automatischen Montieren und Zugreifen auf entfernte Dateisysteme in einer Netzumgebung. Ein virtuelles Dateisystem vereinfacht den Zugriff auf eine virtuelle, logische Speichereinrichtung, wobei das Dateisystem mindestens einen Teil eines entfernten Dateisystems beinhaltet, das von einer weiteren Einrichtung verwaltet wird, die im Computernetz angeschlossen ist. Das Betriebssystem leitet Zugriffsanforderungen von einem Anwendungsprogramm, die die virtuelle, logische Speichereinrichtung bezeichnen, an ein entferntes Zugriffselement zur Verarbeitung um. Wenn nötig ermöglicht das entfernte Zugriffselement das automatische Montieren eines Elementes des entfernten Dateisystems.

ZUSAMMENFASSUNG DER ERFINDUNG

[0028] Spezielle und bevorzugte Aspekte der Erfindung werden in den beigefügten unabhängigen und abhängigen Ansprüchen dargelegt. Eigenschaften der abhängigen Ansprüche können mit denen der unabhängigen Ansprüche nach Bedarf und in anderen Kombinationen als denen, die in den Ansprüchen explizit dargelegt werden, kombiniert werden.

[0029] Zusammengefaßt ist die vorliegende Erfindung ein System und ein Verfahren, die transparenten, globalen Zugriff auf Einrichtungen bzw. Geräte auf einem Computer-Cluster bieten.

[0030] Eine Ausführungsform der Erfindung umfaßt einen gemeinsamen Betriebssystemkern, der auf jedem Knoten, die das Cluster ausmachen, läuft, ein Dateisystem, das auf allen Knoten läuft; eine Gerätetreiberschnittstelle (DDI), die auf jedem Knoten läuft, ein Gerätekonfigurationssystem (DCS), das auf einem der Knoten läuft, eine DCS-Datenbank, auf die das DCS und eine Mehrzahl von Gerätetreibern, die auf jedem Knoten liegt, zugreifen können.

[0031] Jeder der Gerätetreiber verwaltet eine Art von physikalischen Geräten und ihm ist eine eindeutige, vorher festgelegte Major-Nummer zugeordnet. Wenn ein neues Gerät einer bestimmten Art zu einem entsprechenden Knoten hinzugefügt wird, wird eine Zufüge- bzw. Attach-Nachricht zu der DDI dieses Knotens ausgegeben, die Konfigurierungsinformation für das Gerät, das hinzugefügt wird, angibt. Die DDI erzeugt mittels der Konfigurierungsinformation einen physikalischen Namen in dem Namensraum des Dateisystems für das Gerät und einen logischen Namen, der eine symbolische Verknüpfung mit dem physikalischen Namen ist. Der logische Name für das Gerät kann anschließend verwendet werden, um über das Dateisystem auf das Gerät zuzugreifen.

[0032] Als Teil des Erzeugens des logischen Namens setzt das DDI eine Abbildungsanforderung an das DCS ab, um eine globale Minor-Nummer (gmin-Nummer) für das hinzugefügte Gerät anzufordern. Die Abbildungsanforderungs-Nachricht umfaßt neben anderen Dingen die Major-Nummer und mindestens eine Teilmenge der Konfigurierungsinformation.

[0033] Als Reaktion auf die Abbildungsanforderung ist das DCS dafür ausgelegt:
(a) die gmin-Nummer festzulegen,
(b) die gmin-Nummer an das DDI zurückzugeben und
(c) die gmin-Nummer, die Major-Nummer und die Teilmenge der Konfigurierungsinformation zu speichern.

[0034] Die anfordernde DDI bildet dann den logischen Namen und leitet einen dem Gerät zugeordneten dev_t-Wert mittels der zurückgelieferten gmin-Nummer ab und aktualisiert die lokale Geräteinformation, so daß der dev_t-Wert des Gerätes für das Dateisystem zugreifbar ist.

[0035] Durch Bereitstellen eines eindeutigen dev_t-Wertes für alle Geräte und einer Verknüpfung zwischen dem Dateisystem und diesem dev_t-Wert bietet die vorliegende Erfindung ein globales Rahmenwerk, das es ermöglicht, auf Geräte bzw. Einrichtungen an unterschiedlichen Knoten global zugreifen zu können. Das Dateisystem wird modifiziert, um aus diesem System Vorteile zu ziehen, so daß in dem Fall, daß ein Benutzer anfordert, ein bestimmtes Gerät, bezeichnet durch seinen logischen Namen, zu öffnen, der Kern bei dem Dateisystem anfragt, um den dev_t-Wert dieses Gerätes zu ermitteln, und dann bei dem DCS den Ort und die Bezeichnung eines Gerätes mit diesem dev_t-Wert abfragt. Sobald er den Ort und die Bezeichnung des Gerätes erhalten hat, setzt der Kern eine Anforderung zum Öffnen an den Hostknoten für das durch das DCS identifizierte Gerät ab. Auf dem Hostknoten ausgeführte Dateisystemkomponenten, die ein spezielles Dateisystem (SpecFS) umfassen, behandeln die Anforderung zum Öffnen, indem sie an den Kern ein Handle auf ein spezielles Dateiobjekt zurückliefern, das dem gewünschten Gerät zugeordnet ist. Der Kern gibt dann an den anfordernden Benutzer einen auf das Handle abgebildeten Datei-Deskriptor zurück, durch den der Benutzer auf das Gerät zugreifen kann.

[0036] Nach einer bevorzugten Ausführungsform können sich das DCS, das Dateisystem, der Benutzer und das angeforderte Gerät alle auf unterschiedlichen Knoten befinden. Um in dieser Umgebung zu funktionieren, beinhaltet die vorliegende Erfindung ein Proxy-Dateisystem, das es dem Benutzer eines Cluster-Knotens ermöglicht, transparent mit den Dateiobjekten zu kommunizieren, die zusammen mit einem angeforderten Gerät auf einem anderen Knoten liegen.

[0037] Die vorliegende Erfindung kann auch einen Satz von Gerätetreiberobjekten (Device Driver Objects, DSOs) auf jedem Knoten des Clusters beinhalten, von denen jedes eine bestimmte Klasse von Geräten verwaltet. Die betreffenden Geräteklassen fassen die Besonderheit zusammen, mit der einer Anforderung eines Benutzers zum Öffnen eines bestimmten Gerätes von dem transparenten, globalen Gerätezugriffssystem im allgemeinen und dem DCS im besonderen entsprochen werden muß. Nach einer bevorzugten Ausführungsform gibt es vier Geräteklassen: dev_enumerate, dev_node_specific, dev_global und dev_nodebound.

[0038] Die dev_enumerate-Klasse ist Geräten zugeordnet, die mehrere Instanzen an einem bestimmten Knoten haben können, die von ihrem zugeordneten Treiber aufgezählt werden, wenn das jeweilige Geräte hinzugefügt wird (z. B. mehrere SCSI-Platten). Die dev_node_specific-Klasse ist Geräten zugeordnet, von denen es nur eine Instanz pro Knoten gibt (z. B. Kernspeicher) und die folglich von ihren Treibern nicht aufgezählt werden. Die dev_global-Klasse ist für diejenigen Geräte, auf die entweder lokal oder entfernt mittels eines Treibers zugegriffen werden kann, der auf jedem Knoten angesiedelt ist (z. B. Modems und Netzschnittstellen). Die dev_nodebound-Klasse wird für Einrichtungen verwendet, auf die nur mittels eines Treibers auf einem bestimmten Knoten zugegriffen werden kann, und wenn dieser Knoten nicht verfügbar ist, dann von einem Treiber auf einem anderen Knoten (z. B. hochverfügbare Einrichtungen).

[0039] Wenn Klassen verwendet werden, dann beinhaltet die Gerätekonfigurationsinformation, die von dem Treiber an die DDI ausgegeben wird, vorzugsweise die Klasse des Gerätes. Falls verfügbar, bezieht die DDI diese Klasseninformation in seine Abbildungsanforderung an das DCS ein. Beim Empfang einer Klasseninformation enthaltenden Abbildungsanforderung befragt das DCS sein lokales DSO zu dieser Klasse. Dieses DSO ermittelt dann die gminor-Nummer, die dem Gerät, das hinzugefügt wird, zugewiesen werden sollte. Zum Beispiel weist das DSO für die dev_enumerate-Klasse jedem dev_enumerate-Gerät eine gmin-Nummer zu, die über das Cluster hinweg eindeutig ist, weil auf jedes aufgezählte Gerät an einem bestimmten Knoten zugegriffen werden muß. Im Gegensatz dazu weist das DSO für die dev_global-Klasse jedem globalen Gerät denselben gmin-Wert zu, weil es unerheblich ist, an welchem Knoten auf solche Geräte zugegriffen wird. Wie für die

anderen Klassen weist das DSO für die dev_node spezifische Klasse jedem Gerät dieser Klasse denselben gmin-Wert ungleich Null zu und das DSO für die dev_nodebound-Klasse weist jedem Gerät dieser Klasse eine gmin-Nummer zu, die über das Cluster hinweg eindeutig ist.

[0040] Wenn die Klasseninformation von einem Treiber nicht bereitgestellt wird, behandelt die vorliegende Erfindung das entsprechende Gerät, als wenn es von der dev_enumerate-Klasse oder der dev_global-Klasse wäre abhängig davon, ob es ein physikalisches Gerät (dev_enumerate) oder ein Pseudo-Gerät (dev_global) ist.

KURZBESCHREIBUNG DER ZEICHNUNGEN

[0041] Beispielhafte Ausführungsformen der Erfindung werden anschließend nur als Beispiel beschrieben unter Bezug auf die beigefügten Zeichnungen, von denen:

[0042] Fig. 1 ein Blockdiagramm eines Computersystems nach dem Stand der Technik ist, das Komponenten zeigt, die verwendet werden, um Zugriff auf Einrichtungen auf einem einzelnen Computer zur Verfügung zu stellen;

[0043] Fig. 2A ein Blockdiagramm ist, das die Beziehungen zwischen Anwendungen, dem Betriebssystemkern, dem Dateisystem und den Einrichtungen nach dem Stand der Technik zeigt;

[0044] Fig. 2B ein Blockdiagramm ist, das die Beziehungen zwischen logischen Gerätenamen, physikalischen Namen, dem Dateisystem, Gerätetypbezeichnern (dev_t) und Geräten bzw. Einrichtungen nach dem Stand der Technik zeigt;

[0045] Fig. 3 ein Diagramm eines beispielhaften Geräteinformationbaumes (DevInfo-Baumes) ist, der mit den nach dem Stand der Technik verwendeten konsistent ist;

[0046] Fig. 4 ein Flußdiagramm von Operationen ist, die in dem Speicher **104** des Computersystems **100** nach dem Stand der Technik beim Öffnen eines Gerätes durchgeführt werden, wie von einer Anwendung **150** angefordert;

[0047] Fig. 5 ein Blockdiagramm eines Computerclusters ist, in dem die vorliegende Erfindung implementiert werden kann;

[0048] Fig. 6 ein Blockdiagramm von Speicherprogrammen und Datenstrukturen ist, die die vorliegende Erfindung wie in den repräsentativen Knoten **202** und **204** des Clusters von Fig. 5 implementiert ausmachen;

[0049] Fig. 7A ein Flußdiagramm ist, das die Operationen veranschaulicht, durch die das Gerätetreiberschnittstellen-(DDI)-System und das Gerätekonfigurationssystem (DCS) einen geeigneten dev_t-Wert, logischen Namen und physikalischen Namen für ein Gerät einrichten, das dem Knoten **202** hinzugefügt wird;

[0050] Fig. 7B die Beziehung zwischen dem bzw. der logischen Minor-Namen/Nummer, physikalischen Namen und logischen Namen darstellt, die von der vorliegenden Erfindung eingerichtet werden; und

[0051] die Fig. 8A und Fig. 8B Flußdiagramme sind, die Schritte veranschaulichen, die von der vorliegenden Erfindung durchgeführt werden als Reaktion auf eine Anforderung von einer auf einem Knoten **202-1** ausgeführten Anwendung **150**, auf ein Gerät zuzugreifen (öffnen), das sich auf einem Knoten **202-3** befindet.

BESCHREIBUNG DER BEVORZUGTEN AUSFÜHRUNGSFORM

[0052] In Fig. 5 wird ein Biockdiagramm eines Computerclusters **201** gezeigt, in dem die vorliegende Erfindung implementiert werden kann. Der Cluster **201** beinhaltet eine Mehrzahl von Knoten **202** mit zugeordneten Geräten **106** und Anwendungen **150**. Wie in Fig. 1 können die Geräte **106** Hochverfügbarkeitseinrichtungen **112**, Drucker **114**, Kernspeicher **116**, Kommunikationseinrichtungen **118** und Speichereinrichtungen **120** umfassen. Für die Zwecke der vorliegenden Diskussion läuft ein globales Dateisystem **206**, das einen einzigen globalen Dateiraum für alle auf dem Cluster **201** gespeicherten Dateien auf dem Laufenden hält, auf einem der Knoten **202**. Das globale Dateisystem **206** unterstützt mindestens zwei Repräsentationen der Geräte **106**. Die Repräsentation im physikalischen Namensraum (PNS) **305** ist aus dem Kernspeicher zugänglich und entspricht der physikalischen Anordnung des Gerätes **106** auf den entsprechenden Knoten **202**. Die Repräsenta-

tion im logischen Namensraum (LNS) **304** ist eine Version des physikalischen Namensraumes **305** im Benutzerspeicher; d. h. jeder Eintrag in dem logischen Namensraum **304** bildet sich auf einen entsprechenden Eintrag in dem physikalischen Namensraum **305** ab. Die vorliegende Erfindung modifiziert viele Aspekte dieses globalen Dateisystems **206**, um einen transparenten, globalen Zugriff auf die Geräte **106** durch die Anwendungen **150** zu ermöglichen. Der Cluster **201** beinhaltet auch einen Knoten **205**, der ein Gerätekonfigurationssystem (DCS) **208** beheimatet, das eine Schlüsselkomponente einer Ausführungsform der Erfindung ist.

[0053] In anderen Ausführungsformen könnte es eine beliebige Anzahl von globalen Dateisystemen **206** geben, von denen jedes seinen eigenen physikalischen und logischen Namensraum auf dem Laufenden hält. In einer solchen Ausführungsform wird auf eine bestimmte Einrichtung nur durch eines der globalen Dateisysteme **206** und dessen zugeordneten physikalischen und logischen Namensraum zugegriffen.

[0054] Wie oben mit Bezug auf die Fig. 1–Fig. 4 beschrieben erlaubt das frühere Gerätezugriffssystem von Solaris einen transparenten Gerätezugriff nur bei einem einzelnen Computersystem. Bestimmte Aspekte der Art, wie der Stand der Technik die logischen Namen erzeugt, die von dem Dateisystem auf den dev_t-Wert des Gerätes, auf das zugegriffen werden soll, abgebildet werden, sind nicht mit der Ausdehnung des gegenwärtigen Gerätezugriffssystems auf einen Cluster kompatibel. Unter der Annahme, daß die Sätze von Geräten **106-1**, **106-2** jeweils vier SCSI-Plattenlaufwerke enthalten, würde das aktuell verwendete logische Benennungssystem zum Beispiel dazu führen, daß unterschiedliche Laufwerke auf den verschiedenen Knoten **106-1**, **106-2** denselben dev-t-Wert haben. Dies würde es für eine Anwendung **150-1** unmöglich machen, auf ein spezielles der Plattenlaufwerke auf dem Knoten **202-2** transparent zuzugreifen. Es wird nun beschrieben, wie eine Ausführungsform der Erfindung einen solchen transparenten globalen Gerätezugriff zur Verfügung stellt.

[0055] In Fig. 6 werden zusätzliche Details eines Repräsentanten der Knoten **202** und des Knotens **204**, der das DCS **208** beheimatet, abgebildet. Das Dateisystem **206** ist in dieser Figur nicht abgebildet, da es sich nur auf einem bestimmten Knoten **202-2** befindet. Jeder Knoten **202** beinhaltet einen Speicher **230**, in dem Betriebssystem-(OS)-Routinen/Objekte **240** und Datenstrukturen **300** definiert sind. Die OS-Routinen **240** beinhalten einen Betriebssystemkern **242**, ein Proxy-Dateisystem (PxFS) **244**, ein spezielles Dateisystem **258**, ein Gerätetreiber-System (DDI) **270**, eine Menge von Gerätetreiberobjekten (DSO) und Gerätetreiber **280**.

[0056] Wie oben beschrieben, behandelt der Kern **242** Systemaufrufe von den Anwendungen **150** wie etwa Anforderungen, auf den Speicher **230**, das Dateisystem **206** oder Geräte **106** zuzugreifen. Der Kern **242** unterscheidet sich von dem Kern **132** (Fig. 1), da er durch die vorliegende Erfindung modifiziert wurde, um globalen Gerätezugriff zu unterstützen. Das Proxy-Dateisystem (PxFS) **244** basiert auf dem Solaris-PxFS-Dateisystem, ist aber hier wie der Kern **242** modifiziert, um globalen Gerätezugriff zu unterstützen. Das PxFS **244** beinhaltet eine Sammlung von Objekten, die eine Anwendung **150-i** auf einem Knoten **202-i** in die Lage setzen, nahtlos mit dem Dateisystem **206** über verschiedene Knoten **202** hinweg zu interagieren. Die PxFS-Objekte beinhalten PxFS-Clients **246**, PxFS-Server **248**, f_objs (Dateiobjekte) **250**, vnodes (virtuelle Dateiknoten) **252**, snodes (spezielle Dateiknoten) **254** und px_vnodes (Proxy-vnodes) **256**. Jedes dieser Objekte ist in Fig. 6 als optional (opt) bezeichnet, da sie erzeugt werden, wie es von dem PxFS **244** als Reaktion auf Operationen des Dateisystems **206** benötigt wird.

[0057] Das DDI-System **270** (im folgenden als das DDI bezeichnet) ist ebenfalls ähnlich zu dem mit Bezug auf den Stand der Technik (Fig. 1) beschriebenen DDI-System **142**. Das DDI-System **270** ist jedoch in einer Ausführungsform der Erfindung abgewandelt, um mit dem DCS **360** zu interagieren und physikalische und logische Namen zu erzeugen, die mit den Geräten **106** kompatibel sind, auf die auf und von den unterschiedlichen Knoten **202** zugegriffen werden kann. Das DDI **270** beinhaltet eine Hinzufügen-Methode **272**, die jedesmal aufgerufen wird, wenn ein neues Gerät dem lokalen Knoten **202** hinzugefügt wird. Im Gegensatz zu der früheren Hinzufügen-Methode ist die Hinzufügen-Methode **272**. dafür ausgelegt, die Dienste des DCS **360** zu verwenden, um einen global konsistenten physikalischen Namen für jedwedes hinzugefügte Gerät zu erzeugen. Das DDI-System **270** beinhaltet auch eine Sammlung von Verknüpfungsgeneratoren **274**, die eindeutige logische Namen aus den entsprechenden physikalischen Namen erzeugen. Es gibt unterschiedliche Arten von Verknüpfungsgeneratoren für jede unterschiedliche Art von Geräten bzw. Einrichtungen **106**. Daher konstruieren die Hinzufügen-Routine **272** bzw. die Verknüpfungsgeneratoren **274** die physikalischen und logischen Namensräume, die die Geräte **106** auf Kern- bzw. Benutzerebene global sichtbar machen.

[0058] Eine Ausführungsform der Erfindung beinhaltet einen Satz von DSOs **290** auf jedem Knoten des Clusters **200**, von denen jedes eine bestimmte Klasse **312** von Einrichtungen **106** verwaltet. Die entsprechenden Geräteklassen sind ein neuer Aspekt der vorliegenden Erfindung, die die Besonderheit abdecken, mit der einer Benutzeranforderung, ein bestimmtes Gerät **106** zu öffnen, von dem transparenten, globalen Gerätezugriffs-

system im allgemeinen und von dem DCS **372** im besonderen genügt werden muß. In der bevorzugten Ausführungsform gibt es vier Geräteklassen: dev_enumerate **314**, dev_node_specific **316**, dev_global **318** und dev_nodebound **320**; und vier entsprechende DSOs **290**: DSO_enum **292**, DSO_nodespec **294**, DSO_global **296** und DSO_nodebound **298**.

[0059] Die dev_enumerate-Klasse **314** ist Geräten **106** zugeordnet, die mehrere Instanzen an einem bestimmten Knoten **202** haben können, die von ihrem zugeordneten Treiber **280** durchnumeriert werden, wenn das jeweilige Gerät hinzugefügt wird (z. B. mehrere Speichereinrichtungen **120**). Die dev_node_specific-Klasse **316** ist Geräten **106** zugeordnet, von denen es nur eine Instanz pro Knoten gibt (z. B. den Kernspeicher **116**) und die als Folge davon nicht von ihren Treibern **280** durchnumeriert werden. Die dev_global-Klasse **318** ist für jene Einrichtungen **106** vorgesehen, auf die entweder lokal oder entfernt mittels eines Treibers zugegriffen werden kann, der sich auf jedem Knoten befindet (z. B. Kommunikationseinrichtungen **118**). Die dev_nodebound-Klasse wird für Geräte verwendet, auf die nur mittels eines Treibers auf einem bestimmten Knoten zugegriffen werden kann (z. B. HA-Einrichtungen **112**).

[0060] Die Treiber **280** sind ähnlich den Treibern **140**, außer daß sie zusätzliche Konfigurationsinformation für jedes Objekt, das hinzugefügt wird, mitteilen, einschließlich der Geräteklasseninformation **312**, falls verfügbar.

[0061] Die Datenstrukturen **300** beinhalten einen DevInfo-Baum **302** und eine ddi_minor_nodes-Tabelle **306**. Wie viele der OS-Routinen **240** sind die Datenstrukturen **300** ähnlich den gleichnamigen Datenstrukturen **160**, die nach dem Stand der Technik verwendet werden (Fig. 1). Jede enthält jedoch wichtige Unterschiede gegenüber dem Stand der Technik, die das Funktionieren der vorliegenden Erfindung ermöglichen. Insbesondere enthält der DevInfo-Baum **302** zusätzliche Zwischenknoten, die benötigt werden, um Einrichtungen ausgewählter Klassen innerhalb des Clusters **200** zu lokalisieren. Als eine Folge der Änderungen am physikalischen Namensraum **305**, die durch den DevInfo-Baum repräsentiert werden, unterscheidet sich auch der logische Namensraum **304** von dem logischen Namensraum **164** nach dem Stand der Technik. Schließlich enthält die ddi_minor_nodes-Tabelle **306** im Vergleich zu der nach dem Stand der Technik verwendeten ddi_minor_nodes-Tabelle zusätzliche Felder. Zum Beispiel enthält die vorliegende ddi_minor_nodes-Tabelle die Felder global_minor_number, local_minor_number und (Geräte)-Klasse **308**, **310** und **312** (oben beschrieben); die ddi_minor_nodes-Tabelle nach dem Stand der Technik enthielt keines der Felder **308** oder **312**.

[0062] Der Knoten **204** beinhaltet einen Speicher **330**, in dem die OS-Routinen/-Objekte **340** und die Datenstrukturen **370** definiert sind. Die OS-Routinen/-Objekte **340** beinhalten das Gerätekonfigurationssystem (DCS) **360**, eine map_minor-Methode **362** auf dem DCS und einen Satz von DSOs **290**, die identisch zu den bereits beschriebenen sind. Die Datenstrukturen **370** beinhalten eine DCS-Datenbank **372**.

[0063] Das DCS **360**, zu dem es beim Stand der Technik kein Analogon gibt, dient mindestens zwei wichtigen Funktionen. Als erstes arbeitet das DCS **360** mit den DDIs **270**, um globale Nebennummern ("Minor-Nummern") neu hinzugefügten Geräten zuzuweisen, die es ermöglichen, daß diese Geräte global und transparent zugreifbar sind. Als zweites wirkt das DCS **360** mit dem Dateisystem **206** und dem PxFS **244**, Anwendungen **150** in die Lage zu versetzen, auf hinzugefügte Geräte **106** transparent zuzugreifen. Die DCS-Datenbank **372** hält alle wichtigen von dem DCS **372** erzeugten Ergebnisse in persistentem Speicher. Die beiden Aspekte der DCS **360** werden nun anschließend unter Bezug auf die Fig. 7A–B bzw. Fig. 8A–B beschrieben.

[0064] In Fig. 7A ist ein Flußdiagramm abgebildet, das die Operationen darstellt, durch die das DDI-System in einem Knoten **202** und das DCS **360** in dem Knoten **204** einen passenden dev_t-Wert, einen logischen Namen und einen physikalischen Namen für ein Gerät **380**, das zu dem Knoten **202** hinzugefügt wird, einrichten. Die DDIs **270**, die Verknüpfungsgeneratoren **274**, das DCS **360** und Erweiterungen davon fungieren gemeinsam als ein Geräte-Registrator für das Cluster **200**. Die Operationen und Nachrichten sind in derselben Weise wie in **Fig.** 4A angegeben. Bevor die in dem Flußdiagramm wiedergegebenen Operationen beschrieben werden, wird unter Bezug auf Fig. 7B die Beziehung zwischen einigen der Namensräume beschrieben, die von einer Ausführungsform der Erfindung verwaltet werden.

[0065] In Fig. 7B ist ein konzeptionelles Diagramm des Minor-Namens-/Nummern-Raumes **307**, des physikalischen Namensraumes **305** und des logischen Namensraumes **304** abgebildet, die in einer Ausführungsform der Erfindung für einen beispielhaften Cluster, der die beiden Knoten **202-1**, **202-2** enthält, verwendet werden. Wie unten beschrieben weist jedesmal, wenn ein Gerät **106** zu einem Knoten **202** hinzugefügt wird, dessen Treiber ihm eine lokale Minor-Nummer **307**_num und einen Namen **307**_name zu. Das DDI **270** verwendet diese Information, um eine global eindeutige Minor-Nummer zu erzeugen und einen global eindeutigen physikalischen Namen **305**_name für das Gerät **106** zu bilden. Der physikalische Name **305**_name lokalisiert das

Gerät in der Gerätehierarchie des Clusters. Die Verknüpfungsgeneratoren **274** bilden dann den physikalischen Namen **305**_name auf einen global eindeutigen logischen Namen **304**_name ab. Man beachte, daß die DDIs **270-1**, **270-2** und die Verknüpfungsgeneratoren **274-1**, **274-2** gemeinsam allgemeine, globale, physikalische und logische Namensräume **305** bzw. **304** erzeugen. Im Gegensatz dazu erzeugt jeder Treiber einen Minor-Namens/Nummern-Raum nur für seinen Knoten **202**. Daher bildet die Ausführungsform lokale Minor-Namen/Nummern auf globale physikalische und logische Namen ab. Diese globalen Namensräume sind Teil des Dateisystems **206**. Folglich kann eine Anwendung **150** auf irgendeinem Knoten **202** das Dateisystem **206** verwenden, um alle Geräte **106** auf dem Cluster **200** zu sehen und darauf zuzugreifen, als wenn sie sich auf einem einzigen Computer befinden würden. Nachdem die Namensräume beschrieben wurden, die den Rahmen für sie bilden, wird nun eine Ausführungsform der Erfindung unter Bezug auf Fig. 7B beschrieben.

[0066] Wenn das Gerät **106** in Fig. 7A zu dem Knoten **202** hinzugefügt wird, setzt das DDI **270** eine Hinzufügen-Nachricht (**7-1a**) an den Treiber **280** ab. Im Gegenzug setzt der Treiber **280** eine create_ddi_minor_nodes-Nachricht (**7-1b**) für jedes der gerade hinzugefügten Instanz zugeordnete Gerät an das DDI **270** ab. Die create_ddi_minor_nodes-Nachricht (**7-1b**) gibt die Konfiguration des Gerätes **380** einschließlich einer lokalen Minor-Nummer (minor_num) **382** und eines minor_name **384** an, die vom geeigneten Gerätetreiber **280** und einer aus den Klassen **312** ausgewählten device_class **386** zugewiesen werden. Wenn zum Beispiel das Gerät das dritte zu dem Knoten **202** hinzugefügte SCSI-Plattenlaufwerk wäre, könnten minor_num, minor_name und Klasse "3", "a" (was angibt, daß es die erste Scheibe bzw. Slice auf diesem Gerät ist) bzw. "dev_enumerate" sein.

[0067] Als Reaktion auf die create_minor_nodes-Nachricht (**7-1b**) aktualisiert das DDI **270** die ddi_minor_nodes-Tabelle **306**, indem das local_minor_num-Feld **310** gleich dem minor_num-Wert **382** gesetzt wird (**7-2**). Das DDI **270** setzt dann eine dc_map_minor-Nachricht (**7-3**) an das DCS **360** ab, die das DCS **360** auffordert, eine passende globale Minor-Nummer **388** für das Gerät **380** zurückzugeben. Was im vorstehenden Satz mit "passend" gemeint ist, hängt von der Geräteklasse ab. Das bedeutet, dev_enumerate- und dev_nodebound-Geräte erfordern eindeutige globale Minor-Nummern **388** und dev_global- und dev_nodespecific-Geräte erfordern das nicht. Die dc_map_minor-Nachricht (**7-3**) hat drei Felder: (1) "gminor", welches ein Rückgabefeld für die von dem DCS **360** erzeugte globale Minor-Nummer **388** ist; (2) "lminor", welches die von dem Gerätetreiber **280** erzeugte lokale Minor-Nummer **384** enthält; und (3) "class", welches die von dem Gerätetreiber **280** erzeugte Geräteklasse **386** enthält. Als Reaktion auf die map_minor-Nachricht (**7-3**) setzt das DCS **360** eine ähnliche ds_map_minor-Nachricht (**7-4**) an das lokale DSO **290** für die in der Nachricht (**7-3**) angegebene Klasse ab.

[0068] Das DSO **290** bestimmt unter anderem die globale Minor-Nummer (gmin) **388**, die dem Gerät, das hinzugefügt wird, zugewiesen werden sollte. Wie die gmin-Nummer zugewiesen wird, hängt von der Klasse **386** des Gerätes ab. Zum Beispiel weist das DSO **292** für die dev_enumerate-Klasse **314** jedem dev_enumerate-Gerät eine gmin-Nummer **388** zu, die über den Cluster hinweg eindeutig ist, weil auf jedes aufgezählte Gerät an einem bestimmten Knoten zugegriffen werden muß. Im Gegensatz dazu weist das DSO **296** für die dev_global-Klasse **318** jedem dev_global-Gerät dieselbe gmin-Nummer zu, da es unerheblich ist, an welchem Knoten auf ein solches Gerät zugegriffen wird. Wie für die anderen Klassen weist das DSO **294** für die dev_nodespecific-Klasse **316** jedem Gerät dieser Klasse dieselbe gmin-Nummer ungleich Null zu und das DSO **298** weist für die dev_nodebound-Klasse **320** jedem Gerät dieser Klasse eine gmin-Nummer zu, die über den Cluster hinweg eindeutig ist.

[0069] Die DSOs **292**, **298** weisen globale Minor-Nummern zu, indem sie zuerst die DCS-Datenbank **372** befragen, um festzustellen, welche globale Minor-Nummern noch verfügbar sind.

[0070] Die DCS-Datenbank **372** wird in persistentem Speicher gehalten und enthält für alle Geräte **106** in dem Cluster **200** Felder für die Major-Nummer **390**, die globale Minor-Nummer **388**, die interne (oder lokale) Minor-Nummer **382** und die Geräte-Server-Id **392** (die die Server-Klasse **386** und den numerischen Wert **394** beinhaltet). Der Minor-Name, die Major-Nummer, die globale Minor-Nummer und die lokale Minor-Nummer wurden bereits beschrieben. Der numerische Wert **394** bezeichnet den Knoten **202**, der der Server für das Gerät, das hinzugefügt wird, ist. Diese Information ist für die dev_global- und dev_nodespecific-Geräte optional, da die Kennung eines Servers für die erste Klasse unerheblich ist, und für den zweiten Fall dieselbe wie die Lokation des Knotens ist, welcher Knoten auch immer auf das Gerät zugreifen möchte. Ein Beispiel der DCS-Datenbank **272** ist in Tabelle 1 abgebildet.

Tabelle 1

Gerät (kein Feld)	Major 390	globale Minor 388	interne Minor 382	Geräte-Server-Id 392:	
				Server Klasse 386	numerischer Wert 394
tcp	42	0	0	dev_global	0
kmem	13	12	12	dev_node_spec	0
disk c2t0d0s0	32	24	24	dev_enum	node id
kmem	13	1	12	dev_enum	node 0 id
kmem	13	2	12	dev_enum	node 1 id
kmem	13	3	12	dev_enum	node 2 id
kmem	13	4	12	dev_enum	node 3 id
HA-Geräte	M	X1	X1	dev_nodebound	id

[0071] Die erste Zeile von Tabelle 1 zeigt einen Eintrag für eine tcp-Schnittstelle. Eine tcp-Schnittstelle ist ein dev_global-Gerät, da auf sie von jedem Knoten **202** in dem Cluster **200** zugegriffen werden kann. Das tcp-Gerät hat eine Major-Nummer von **42**, welches der allen tcp-Treibern zugeordnete Wert ist. Man beachte, daß ihre globalen und lokalen Minimalwerte **388**, **382** und der numerische Serverwert **394** (d. h. node_id) auf Null gesetzt sind. Das liegt daran, daß es unerheblich ist, von welchem Knoten auf die tcp-Schnittstelle zugegriffen wird. Folglich gibt es nur einen tcp-Eintrag in der DCS-Datenbank für den gesamten Cluster **200**. Der zweite Eintrag in Tabelle 1 ist für ein Kernspeicher-Gerät, auf welches per Voreinstellung lokal zugegriffen wird. Aus diesem Grund ist es von der dev_nodespecific-Klasse. Die Major-Nummer **13** ist dem kmem-Gerätetreiber zugeordnet. Das kmem-Gerät hat einen numerischen Wert **394** von Null, da auf kmem-Geräte nicht an irgendeinem bestimmten Server zugegriffen wird, und identische globale und lokale Nebenwerte (12) ungleich Null. Das ist der Fall, da für dev_nodespecific-Geräte das DCS **360** einfach eine globale Minor-Nummer zuweist, die mit der lokalen Minor-Nummer identisch ist. In dem vorliegenden Beispiel gibt es nur einen kmem-Eintrag von der dev_nodespecific-Vielfalt in der DCS-Datenbank **372**, da es keinen Bedarf gibt, zwischen den auf den jeweiligen Knoten **202** gelegenen kmem-Geräten zu unterscheiden.

[0072] Der dritte Eintrag ist für eine SCSI-Platte c0t0d0t0, dessen SCSI-Treiber die Major-Nummer **32** hat. Das DCS **360** hat dem SCSI-Gerät eine globale Minor-Nummer **388** zugewiesen, die mit seiner lokalen Minor-Nummer **382** (24) übereinstimmt, da es keine anderen SCSI-Geräte gibt, die in der DCS-Datenbank **372** repräsentiert werden. Wenn jedoch ein anderes SCSI-Gerät c0t0d0t0 an einem anderen Knoten mit derselben lokalen Nummer (24) registriert würde, würde das DCS **360** diesem SCSI eine unterschiedliche globale Nummer zuweisen, vielleicht 25. Um SCSI-Geräte mit denselben lokalen Nummern zu unterscheiden, enthält die DCS-Datenbank **372** die vollständige Serverinformation. In diesem Fall ist der numerische Wert **394** auf die hostid von Server **202** gesetzt.

[0073] Die Einträge vier bis sieben sind für vier Kernspeicher-Geräte, die als dev_enumerate-Geräte registriert sind. In der bevorzugten Ausführungsform können jedesmal, wenn ein dev_nodespecific-Gerät registriert wird, zusätzliche Einträge in der DCS-Datenbank **372** für alle Knoten **202** in dem Kern erzeugt werden, was einem Benutzer ermöglicht, auf ein dev_nodespecific-Gerät auf einem anderen als dem lokalen Knoten zuzugreifen. Folglich kann das DCS **260** unter der Annahme, daß es vier Knoten **202-1**, **202-2**, **202-3** und **202-4** gibt, ein Kernspeicher-Gerät von der dev_enumerate-Klasse für jeden dieser Knoten registrieren. Wie bei anderen dev_enumerate-Geräten wird jedem kmem-Gerät eine eindeutige globale Nummer zugewiesen. Die dev_enumerate-Information würde nicht verwendet werden, wenn ein Benutzer eine generische Anforderung zum Öffnen eines Kernspeicher-Gerätes absetzt (z. B. open(/devices/kmem)). Die dev_enumerate-Information würde verwendet werden, wenn ein Benutzer eine spezifische Anforderung zum Öffnen eines Kernspeicher-Gerätes absetzt. Zum Beispiel ermöglicht die Anforderung open(/devices/kmem0) einem Benutzer, das kmem-Gerät auf dem Knoten 0 zu öffnen.

[0074] Der letzte Eintrag zeigt, wie ein generisches Hochverfügbarkeits-(HA)-Gerät in der DCS-Datenbank **372** dargestellt wird. Die Major-Nummer **390**, die globale Minor-Nummer und die lokale Minor-Nummer werden aus den Werten M, X1 und X1 genommen, die in der map_minor_nodes-Nachricht geliefert werden. Der numerische Wert **394** wird auf die Id des Gerätes gesetzt, das an einen bestimmten Knoten gebunden ist. Diese "Id" ist keine Knoten-Id. Vielmehr wird die Id für jeden HA-Dienst eindeutig für den Cluster **200** erzeugt.

[0075] Sobald die globale Minor-Nummer **388** für das Gerät **380** ermittelt ist, aktualisiert das entsprechende

DSO **290** die DCS-Datenbank **372** mit der neuen Information (**7-5**) und gibt die globale Minor-Nummer **388** an das DCS **360** zurück (**7-6**). Das DCS **372** gibt dann die globale Minor-Nummer **388** an das DDI **270** zurück (**7-7**), welches die ddi_minor_nodes-Tabelle **306** (**7-9**), den logischen Namensraum **304**, den physikalischen Namensraum **305** und den dev_info-Baum **302** aktualisiert (**7-9**). Das DDI **270** aktualisiert die ddi_minor_nodes-Tabelle **306**, indem es die neue globale Minor-Nummer **388** dort hineinschreibt. Die Aktualisierung der Namenräume 304/305 ist komplizierter und wird nun beschrieben.

[0076] Zuerst fügt das DDI **270** einen neuen Blattknoten zu dem DevInfo-Baum **302** hinzu, dessen Struktur gegenüber der zuvor mit Bezug auf Fig. 3 beschriebenen geändert wurde, um gleich unter dem "/devices"-Knoten eine zusätzliche Ebene von "/hostid"-Knoten zum Repräsentieren der Stellen im Cluster, an denen dev_enumerate hinzugefügt werden, einzufügen. Man beachte, daß jeder Knoten **202** seinen eigenen DevInfo-Baum **270** hat, der die Geräte an diesem Knoten repräsentiert. Wie durch den physikalischen Namensraum dargestellt, verschmilzt jedoch die Zusammenfassung der DevInfo-Bäume mit den zusätzlichen /hostid-Knoten zu einer einzigen Darstellung. (z. B. könnte ein typischer physikalischer Name mit der Zeichenkette /devices/hostid/... beginnen). Jedem Gerät ist auf der Blattebene auch seine globale Minor-Nummer **388** zugeordnet, nicht seine lokale Minor-Nummer **382**. Wo es darauf ankommt (d. h. für dev_enumerate-Geräte), wird der dev_t-Wert jedes Blattknotens des DevInfo-Baumes **302** von der globalen Minor-Nummer **388** des entsprechenden Gerätes und der Major-Nummer **390** seines Treiber abgeleitet. Zum Beispiel wird der physikalische Pfad zu einer SCSI-Platte an einem Knoten **202-x** mit einer globalen Minor-Nummer GN, einem Minor-Namen MN und Treiber sd@addry in der vorliegenden Erfindung repräsentiert als:
/devices/node_202-x/iommu@addr/sbus@addr/esp@addr/sd@addry:MN.

[0077] Dieser physikalische Name entspricht dem physikalischen Namen der UFS-Datei **170** (Fig. 2B), die Konfigurierungsinformation für das gegebene Gerät enthält, wobei sie in ihren Attributen den von den Major- und globalen Minor-Nummern abgeleiteten dev_t-Wert beinhaltet.

[0078] Die Verknüpfungsgeneratoren **274** der vorliegenden Erfindung leiten einen logischen Namen für das Gerät (und für das entsprechende UFS) von mindestens einem Teil des DevInfo-Pfades und des Minor-Namens ab, der von dem Treiber geliefert wird, modifiziert gemäß der von dem DCS zurückgelieferten globalen Minor-Nummer.

[0079] Zum Beispiel angenommen, daß der Knoten **202-1** eine SCSI-Platte mit vier Abschnitten hat, denen ursprünglich von ihrem Treiber die Minor-Namen a-d und die Minor-Nummern 0-3 zugewiesen sind, und der Knoten **202-2** eine SCSI-Platte mit sechs Abschnitten hat, denen die Minor-Namen a-f und die Minor-Nummern 0-5 zugewiesen sind. Es sei angenommen, daß das DCS **360** für die erste SCSI-Platte die globalen Minor-Nummern 0-3 und für die zweite SCSI-Platte die globalen Minor-Nummern **4-9** zurückliefert, wenn diese Geräte hinzugefügt werden. Mittels dieser globalen Minor-Nummern erzeugen die DDIs **270** physikalische Namen (unten beschrieben), und die Verknüpfungsgeneratoren **274** verwenden die DDIs **270** zum Erzeugen von logischen Namen, die auf die physikalischen Namen wie folgt abgebildet werden:

Minor-Name vom Treiber **280**	logischer Name von den Verknüpfungsgeneratoren **274**
a (Knoten **202-1**)	/dev/dsk/c0t0d0s0
b "	/dev/dsk/c0t0d0s1
c "	/dev/dsk/c0t0ds2
d "	/dev/dsk/c0t0d0s3
a (Knoten **202-2**)	/dev/dsk/c1t0d0s0
b "	/dev/dsk/c1t0d0s1
...	
f "	/dev/dsk/c1t0d0s5

[0080] Die den Knoten **202-1** und **202-2** zugewiesenen logischen Namen haben unterschiedliche Cluster-Werte (der cx-Teil der logischen Namenszeichenkette cxt0d0sy, wobei "x" und "y" Variablen sind). Das liegt daran, daß sich die logischen Name auf physikalische Gerätenamen abbilden, und in einem Cluster Geräten auf unterschiedlichen Knoten unterschiedlichen Steuerungen zugeordnet sind. Zum Beispiel wird die Steuerung auf Knoten **202-1** als c0 und die Steuerung auf Knoten **202-2** als c1 dargestellt.

[0081] Die DDIs **270** erzeugen den physikalischen Namensraum **305** mittels derselben gmin-Information und erzeugen eine Abbildung zwischen logischen Namen und physikalischen Namen, die Dateien bezeichnen, deren Attribute die dev_t-Werte für die entsprechenden Geräte enthalten. Für das obige Beispiel wird der logische

Namensraum **304** und die Abbildung des logischen Namensraumes auf den physikalischen Namensraum wie folgt aktualisiert (man beachte, daß addr für irgendeine Adresse steht):

logischer Name	physikalischer Name vom DevInfo-Baum **302**
/dev/dsk/c0t0d0s0	/devices/node_202-1/iom-mu@addr/sbus@addr/esp1@addr/sd@0:a
/dev/dsk/c0t0d0s1	" /esp1@addr/sd@0:b
/dev/dsk/c0t0d0s2	" /esp1@addr/sd@0:c
/dev/dsk/c0t0d0s3	" /esp1 @addr/sd@0:d
/dev/dsk/c1t0d0s0	/devices/node_202-2/iom-mu@addr/sbus@addr/esp1@addr/sd@0:minor
/dev/dsk/c1t0d0s1	" /esp1@addr/sd@0:r
...	
/dev/dsk/c1t0d0s2	" /esp1@addr/sd@0:f
/dev/dsk/c0t0d0s5	" /esp1@addr/sd@0:i

[0082] Das gerade dargestellte Beispiel zeigt, daß die DDIs **270** logische und physikalische Namen für dev_enumerate-Geräte erzeugen, wobei SCSI-Geräte Mitglied dieser Klassen sind. Kurz zusammengefaßt erfordern die Regeln zum Benennen von dev_enumerate-Geräten, daß jede von einem bestimmten Treiber (z. B. sd) durchnumerierte Instanz eine eindeutige globale Minor-Nummer haben muß, die, wenn sie mit der Major-Nummer ihres Treibers kombiniert wird, einen entsprechenden eindeutigen dev_t-Wert bildet. Diese Regeln geben auch vor, daß der jeder Instanz zugeordnete physikalische Name die hostid dieser Instanz und die globale Minor-Nummer der Instanz zusätzlich zu anderen traditionellen physikalischen Pfadinformationen enthalten muß. Die Regeln zum Benennen der anderen Geräte von anderen Klassen sind ähnlich zu den oben für die dev_enumerate-Klasse beschriebenen.

[0083] Insbesondere weist das DDI **270** einem dev_nodespecific-Gerät einen logischen Namen der Form /dev/device_name und einen physikalischen Namen der folgenden Form zu: /devices/pseudo/driver@gmin:device_name, wobei device_name der Name **384** ist, pseudo anzeigt, daß die Geräte dieses Typs Pseudo-Geräte sind, driver die Id des entsprechenden Treibers ist und @gmin:device_name die globale Nummer **388** und den Gerätenamen **384** des dev_nodespecific-Gerätes angibt. Zum Beispiel könnten der logische und der physikalische Name eines Kernspeicher-Gerätes /dev/kmem bzw. /devices/pseudo/mm@12:kmem sein. Wie oben erwähnt, kann einem kmem-Gerät auch ein logischer Name gegeben werden, der es ermöglicht, auf einem speziellen Knoten darauf zuzugreifen. Zum Beispiel kann das DDI **270** den logischen Namen /dev/kmem0 auf den physikalischen Namen /devices/hostid0/pseudo/mm@0:kmem abbilden.

[0084] Für die dev_global-Klasse identifiziert jeder von dem DDI erzeugte logische Name einen gemeinsamen physikalischen Pfad, der von dem Dateisystem zu irgendeinem Gerät in dem Cluster **200** aufgelöst wird. Logische Namen für diese Geräte sind von der Form /dev/device_name und werden auf physikalische Namen der folgenden Form abgebildet: /devices/pseudo/clone@gmin:device_name, wobei device_name der Name **384** ist, der spezifisch für den Treiber ist, pseudo anzeigt, daß die Geräte dieses Typs Pseudo-Geräte sind, clone anzeigt, daß das Gerät klonierbar ist und @gmin:device_name die globale Nummer **388** und den Gerätenamen **384** des dev_global-Gerätes angibt. Zum Beispiel könnte das tcp-Gerät von Tabelle 1 einen logischen Name /dev/tcp und einen physikalischen Name /devices/pseudo/clone@0:tcp haben. Man beachte, daß die Ausführungsform der Erfindung nicht zuläßt, daß irgendeines der dev_global-Geräte unterscheidbar gemacht wird wie in dem oben beschriebenen Fall der kmem-Geräte. Das bedeutet, alle dev_global-Geräte sind ununterscheidbar.

[0085] Ein Vorzug des klassenbasierten Benennungssystems einer Ausführungsform der Erfindung ist, daß es mit herkömmlicher Software, die für frühere Versionen von Solaris ausgelegt ist, kompatibel ist. Zum Beispiel könnte ein herkömmliches Programm eine Anforderung open(/dev/kmem) absetzen, in welchem Fall eine Version von Solaris, die die vorliegende Erfindung einbezieht, ein Handle zu dem lokalen kmem-Gerät zurückliefert. Ähnliche Ergebnisse werden für dev_global- und dev_enumerate-Geräte geliefert. Es gab beim Stand der Technik kein Konzept bzw. keinen Ansatz für dev_nodebound-Geräte.

[0086] Nachdem beschrieben wurde, wie das DDI **270** und das DCS **360** einen konsistenten globalen Namensraum bilden, in dem auf unterschiedliche Klassen von Geräten auf unterschiedlichen Knoten des Clusters **200** zugegriffen werden kann, werden nun unter Bezug auf die **Fig. 8A** und **Fig. 8B** die Schritte beschrieben, die von einer Ausführungsform der Erfindung verwendet werden, um auf eine Öffnen-Anforderung für ein Gerät

auf einem anderen Knoten zu antworten.

[0087] In den Fig. 8A und Fig. 8B werden Flußdiagramme der von einer Ausführungsform der Erfindung als Reaktion auf eine Anforderung (**8-1**) durchgeführten Schritte dargestellt, wobei die Anforderung von einer auf einem Knoten **202-1** ausgeführten Anwendung **150** abgesetzt wird, um auf ein Gerät **106-2** (Fig. 8B) zuzugreifen (es zu öffnen), das sich an einem Knoten **202-3** befindet. In diesem Beispiel befinden sich das Dateisystem **206** und das DCS **360** auf den Knoten **202-2** bzw. **204**. Die Anwendung **150** setzt eine Öffnen-Anforderung auf den logischen Namen des Gerätes an den Kern **242** ab. Der Kern **242** fragt dann beim Dateisystem **206** an, um den dev_t-Wert des Gerätes festzustellen. Weil das Dateisystem auf einem anderen Knoten als der Kern **242** liegt, ist dies ein Mehrschritt-Prozeß, der die Verwendung eines Proxy-Dateisystems PxFS involviert, von dem die meisten Aspekte bereits durch aktuelle Versionen von Solaris definiert sind. Die Ausführungsform modifiziert jedoch solche Elemente des Proxy-Dateisystems wie etwa die PxFS-Clients **246** und die PxFS-Server **248**, um Interaktionen mit dem DCS **360** zu unterstützen, für die es kein Analogon in früheren Versionen von Solaris gibt. Die Interaktionen zwischen dem PxFS-Client **246**, dem PxFS-Server **248** und dem Dateisystem **206** werden nun kurz beschrieben.

[0088] Ein Objekt wie der Kern **242**, das Zugriff auf das Dateisystem **206** benötigt, setzt zunächst die Zugriffsanforderung an seinen lokalen PxFS-Client **246** ab. Der PxFS-Client hält eine Referenz auf den PxFS-Server **248**, der bei dem Dateisystem **206** angesiedelt ist. Diese Referenz setzt den PxFS-Client **246** in die Lage, die Anforderung des Kerns an das Dateisystem **206** über den PxFS-Server **248** zu kommunizieren. Das Dateisystem **206** führt den angeforderten Zugriff aus, erzeugt ein vnode-Objekt **252**, das die angeforderte Datei repräsentiert, und gibt eine Referenz auf das vnode-Objekt **252** an den PxFS-Server **248** zurück. Weil die Knoten **202-1** und **202-2** verschiedene Adreßräume sind, ist die Referenz auf den vnode **252** für den PxFS-Client **246** und den Kern **242** in dem Knoten **202-1** nutzlos. Folglich erzeugt der PxFS-Server **248** ein Datei-Transport-Objekt (f_obj) **250**, das mit dem vnode **252** verknüpft ist, und gibt eine Referenz auf das f_obj **250** an den PxFS-Client **246** zurück. Beim Empfang der f_obj-Referenz erzeugt der PxFS-Client **246** einen Proxyvnode (px_vnode) **256**, der mit dem f_obj **250** verknüpft ist. Der Kern **242** kann dann auf die Dateiinformation zugreifen, die durch den vnode **252** repräsentiert wird, indem er einfach auf den lokalen px_vnode **256** zugreift.

[0089] Mittels dieses Mechanismus' setzt der Kern **242** eine Nachschlage- bzw. Lookup-Nachricht (**8-2**) auf den logischen Namen des zu öffnenden Gerätes an den PxFS-Client **246** ab, der eine ähnliche Lookup-Nachricht (**8-3**) an den PxFS-Server **248** weiterleitet. Der PxFS-Server **248** setzt an das Dateisystem **206** eine lookup(logical_name), get_vnode-Nachricht (**8-4**) ab, die das Dateisystem **206** auffordert, über eine logische symbolische Verknüpfung den logical_name auf den zugehörigen physical_name abzubilden und eine Referenz auf einen v_node **252** zurückzugeben, die die durch diesen physical_name bezeichnete UFS-Datei repräsentiert. Wenn sich der physical_name auf ein Gerät bezieht wie in dem vorliegenden Beispiel, beinhalten die Attribute des Gerätes den eindeutigen dev_t-Wert des Gerätes. Wie oben beschrieben gibt das Dateisystem **206** daraufhin den vnode an den PxFS-Server **248** zurück (**8-5**), und der PxFS-Server **248** erzeugt ein entsprechendes f_obj **250** und gibt die Referenz auf das f_obj **250** an den PxFS-Client **246** zurück (**8-6**). Der PxFS-Client **246** erzeugt dann einen px_vnode **256**, dessen Attribute die dev_t-Information für das angeforderte Gerät beinhalten, und übergibt die Referenz auf den px_vnode **256** an den Kern **242** (**8-7**). Zu diesem Zeitpunkt setzt der Kern **242** eine Öffnen-Nachricht (**8-8**) für den px_vnode **246** an den PxFS-Client **246** ab. Auf den Empfang dieser Nachricht hin ermittelt der PxFS-Client **246** aus den Attributen des px_vnode, die einen dev_t-Wert beinhalten, daß der entsprechende vnode **252** ein Gerät repräsentiert und daß die Öffnen-Nachricht daher von dem DCS **360** behandelt werden muß. Wenn der px_vnode **256** keinen dev_t-Wert beinhaltet, würde der PxFS-Client **246** die Öffnen-Anforderung (**8-8**) durch andere Kanäle erfüllen. Wie in früheren Versionen von Solaris implementiert führt der PxFS-Client kein Testen auf dev_t-Werte durch, da Geräte nur lokal zugänglich sind.

[0090] Weil der px-vnode **256** einen dev_t-Wert **430** beinhaltet, setzt der PxFS-Client **246** eine Auflösen-Nachricht (**8-9**) an das DCS **360** für das zu dem dev_t gehörige Gerät ab. Wie das DCS **360** diese Anforderung behandelt, wird nun unter Bezug auf Fig. 8B beschrieben.

[0091] Mit Bezug auf Fig. 8B sieht das DCS **360** als Reaktion auf die resolve(dev_t)-Nachricht (**8-9**) in der DCS-Datenbank **372** nach, um die Lage und die Kennung des Gerätes zu ermitteln, das diesem dev_t-Wert entspricht. Konsistent mit den vorhergehenden Diskussionen der Geräteklassen **312** wird auf Geräte der dev_enumerate- oder dev_nodebound-Klassen auf einem bestimmten Knoten zugegriffen, dessen Lage in dem numerischen Wertefeld **394** der DCS-Datenbank **372** angegeben ist. Im Gegensatz dazu wird auf Geräte der dev_global- oder dev_nodespecific-Klassen auf dem fokalen Knoten der anfordernden Anwendung zugegriffen. Sobald es die Lage des zu öffnenden Gerätes ermittelt hat, gibt das DCS **360** an den PxFS-Client **246**

eine Referenz (DSO_ref) auf das DSO **290** zurück (**8-10**), das die Geräteklasse verwaltet, zu der das angeforderte Gerät gehört und lokal zu dem Knoten ist, der das angeforderte Objekt beheimatet. In dem vorliegenden Beispiel würde unter der Annahme, daß das angeforderte Gerät **106-2** von der dev_enumerate-Klasse ist und auf dem Knoten **202-3** beheimatet ist, das zurückgelieferte DSO_ref auf das DSO_enum-Objekt **292** auf dem Knoten **202-3** verweisen.

[0092] Nach Empfang der Nachricht (**8-10**) setzt der PxFS-Client **246** eine get_device_fobj-Anforderung für das Gerät **106-2** an das referenzierte DSO **292** ab (**8-11**). Im Gegenzug setzt das DSO **292** eine create_specvp()-Nachricht (**8-12**) ab, die das SpecFS **410** auf dem Knoten **202-3** auffordert, den snode für das Gerät **106-2** zu kreieren und zurückzugeben (**8-13**). Das DSO **292** fordert daraufhin die f_obj-Referenz auf den snode von dem PxFS-Server **248-2** an (**8-14a**), der das angeforderte f_obj zurückliefert (**8-14b**). Das DSO **292** gibt dann die f_obj-Referenz auf den snode an den PxFS-Client **246** zurück (**8-15**). Der Client **246** setzt dann eine Öffnen-Anforderung (**8-16**) auf dieses f_obj ab, die über den PxFS-Server **248-2** zu dem SpecFS **410** geht (**8-17**).

[0093] Das SpecFS **410** versucht dann, das Gerät **106-2** zu öffnen. Abhängig vom Ergebnis der Öffnen-Operation gibt das SpecFS **410** eine Status-Nachricht (**8-18**) zurück, die entweder Erfolg oder Fehlschlag anzeigt. Wenn das Öffnen erfolgreich war, enthält die Status-Meldung (**8-18**) auch eine Referenz auf den geöffneten snode **432**. Auf den Empfang von "Erfolg" in der Status-Meldung (**8-18**) hin erzeugt der PxFS-Server **248-2** das f_obj **250-2** für den geöffneten v-node **252-2** und gibt ihn an den PxFS-Client **246** zurück (**8-19**), der einen px_vnode **256-2** erzeugt, der über Knoten hinweg mit dem f_obj **250-2** verknüpft ist. Als abschließenden Schritt in der Geräte-Öffnen-Operation gibt der PxFS-Client den px_vnode **256-2** an den Kern **242** zurück (**8-20**), der einen entsprechenden Dateideskriptor (fd) **434** im Benutzerraum kreiert. Der Kern **242** gibt diesen Dateideskriptor an die Anwendung **150-1** zurück (**8-21**), die dann den Dateideskriptor **434** verwenden kann, um direkt (d. h. über den Kern **242**, den PxFS-Client **246** und den px_vnode) mit dem Gerät **106-2** zu interagieren.

[0094] Während die vorliegende Erfindung unter Bezug auf einige spezielle Ausführungsformen beschrieben wurde, ist die Beschreibung eine Veranschaulichung der Erfindung und ist nicht als Einschränkung der Erfindung auszulegen. Verschiedene Abwandlungen bieten sich für Fachleute auf dem Gebiet an, ohne daß dadurch der Geltungs- bzw. Anwendungsbereich der Erfindung verlassen wird.

Patentansprüche

1. System, welches dafür ausgelegt ist, einen globalen Zugriff auf physikalische Geräte zu gewährleisten, welche auf einem Computercluster angeordnet sind, welches eine Mehrzahl von Knoten aufweist, wobei das System aufweist:
ein globales Dateisystem (**206**),
ein Gerätekonfigurationssystem (DCS) (**360**),
wobei das globale Dateisystem dafür ausgelegt ist, daß es auf eine Anforderung nach Zugriff auf ein solches physikalisches Gerät, die von einem der Knoten ausgegeben wird, reagiert, indem es eine DSO-Handhabe von dem DCS anfordert,
zumindest ein Geräteserverobjekt (DSO) (**290**),
wobei das DCS dafür ausgelegt ist, daß es in Reaktion auf die Anforderung vom dem globalen Dateisystem eine Identität eines ersten DSO bestimmt, welches zu dem angeforderten physikalischen Gerät gehört,
dadurch gekennzeichnet, daß das System weiterhin ein Proxy-Dateisystem (**246**, **248**) aufweist, wobei das DCS dafür ausgelegt ist, zu dem Proxy-Dateisystem eine Referenz auf das erste DSO zu tiefem,
wobei das Roxy-Dateisystem dafür ausgelegt ist, einen Dateideskriptor für den anschließenden Gebrauch beim Zugriff auf das angeforderte physikalische Gerät bereitzustellen.

2. System nach Anspruch 1, wobei das DCS auf einem der Knoten angesiedelt ist und wobei das System weiterhin aufweist:
einen gemeinsamen Betriebssystemkern, der auf jedem der Knoten in dem Computercluster läuft,
eine Gerätetreiberschnittstelle (DDI) (**270**), die auf jedem der Knoten läuft, und
eine Mehrzahl von Gerätetreibern, die auf jedem der Knoten angeordnet sind, wobei jeder der Gerätetreiber dafür ausgelegt ist, einen Typ eines physikalischen Gerätes zu verwalten und jedem dieser Gerätetreiber eine eindeutige größere Zahl zugeordnet ist,
jeder Gerätetreiber dafür ausgelegt ist, dann, wenn ein neues Gerät eines passenden Typs an dem entsprechenden Knoten angebracht wird, eine Anschlußnachricht an die DDI auszugeben, welche eine lokale Kennung (locid) des neu angebrachten Gerätes anzeigt,
wobei die DDI dafür ausgelegt ist, in Reaktion auf die Anschlußnachricht eine Plananforderung an das DCS

nach einer eindeutigen, globalen kleineren (gmin)-Zahl für das angeschlossene Gerät auszugeben, wobei die Plananforderung die größere Zahl und die locid des angeschlossenen Gerätes anzeigt,
und das DCS dafür ausgelegt ist, in Reaktion auf die Plananforderung (a) die gmin-Zahl zu bestimmen, (b) die gmin-Zahl an die DDI zu liefern und (c) die gmin-Zahl, die größere Zahl und die lokale Kennung zu speichern,
und wobei die DDI dafür ausgelegt ist, die gmin-Zahl, die von dem DCS bereitgestellt wird, und die größere Zahl, dem angeschlossenen Gerät zuzuordnen, so daß das angeschlossene Gerät in Reaktion auf eine Anforderung zum Öffnen des angeschlossenen Gerätes von dem Dateisystem aus zugänglich ist.

3. System nach Anspruch 1, wobei das DCS, das globale Dateisystem und das angeforderte Gerät sich jeweils auf verschiedenen Knoten befinden und das Proxy-Dateisystem einen Proxy-Dateisystem-Klienten (**246**) auf einem ersten Knoten und einen Proxy-Dateisystem-Server (**248**) auf einem zweiten Knoten aufweist, welche ermöglichen, daß Anwendungen auf dem ersten Knoten transparent mit Dateiobjekten kommunizieren, die gemeinsam mit dem angeforderten Gerät auf dem zweiten Knoten angeordnet sind.

4. System nach Anspruch 1, wobei das zumindest eine DSO einen Satz von Geräteserverobjekten auf jedem Knoten des Clusters aufweist, von welchem jeder eine entsprechende Geräteklasse verwaltet.

5. System nach Anspruch 4, wobei die Geräteklasse ein Mitglied eines Satzes von Geräteklassen ist, welcher zumindest eine der folgenden umfaßt:
"dev_enumerate", für die Kennzeichnung von Geräten mit zumindest einem Auftreten, welches durch einen besonderen Treiber verwaltet wird, wobei jedes Auftreten, das durch den besonderen Treiber auf einem bestimmten Knoten verwaltet wird, individuell numeriert wird,
"dev_nodespecific", für die Kennzeichnung von Geräten, die an jedem Knoten verfügbar sind und auf welche lokal zugegriffen wird, und die eine eins-zu-eins-Beziehung mit dem Geräteverwaltungstreiber auf jedem Knoten haben,
"dev_global", um Geräte zu kennzeichnen, auf welche von derartigen Gerätetreibern von irgendeinem derartigen Knoten aus zugegriffen werden kann und
"dev_nodebound", welches Geräte kennzeichnet, auf welche durch einen Treiber auf einen bestimmten Knoten zugegriffen werden kann, und welche eine eins-zu-eins-Beziehung mit dem Gerätetreiber haben.

6. Verfahren, welches dafür ausgelegt ist, einen globalen Zugriff auf physikalische Geräte zu gewähren, welche auf einem Computercluster angeordnet sind, welches eine Mehrzahl von Knoten aufweist, wobei das Verfahren die Schritte aufweist.
Reagieren eines globalen Dateisystems (**206**) auf eine Zugriffsanforderung für den Zugriff auf ein solches physikalisches Gerät, welche von einem der Knoten (**202**) ausgegeben wird, in dem eine DSO-Handhabe von einem Gerätekonfigurationssystem (DCS) (**208**) angefordert wird,
wobei das DCS in Reaktion auf die Zugriffsanforderung von dem globalen Dateisystem eine Identität eines ersten Geräteserverobjektes (DSO) (**290**) bestimmt, welche der Anforderung nach dem physikalischen Gerät zugeordnet ist, dadurch gekennzeichnet, daß das Verfahren weiterhin die Schritte aufweist, daß an ein Proxy-Dateisystem (**246**, **248**) eine Referenz bzw. Bezugnahme auf das erste DSO bereitgestellt wird,
wobei das Proxy-Dateisystem einen Dateideskriptor für den nachfolgenden Gebrauch beim Zugriff auf das angeforderte physikalische Gerät liefert.

7. Verfahren nach Anspruch 6, welches weiterhin die Schritte aufweist:
jeder aus einer Mehrzahl von Gerätetreibern gibt, wenn ein neues Gerät eines geeigneten Typs an einen entsprechenden Knoten angeschlossen wird, eine Anschlußnachricht an eine am selben Platz angeordnete Gerätetreiberschnittstelle (DDI) (**270**) aus, welche eine lokale Kennung (locid) des neu angeschlossenen Gerätes anzeigt, wobei jeder der Gerätetreiber dafür ausgelegt ist, einen Typ eines physikalischen Gerätes zu verwalten und jedem der Gerätetreiber eine eindeutige größere Zahl zugeordnet ist,
in Reaktion auf die Anschlußnachricht ausgegeben einer Plananforderung an das DCS nach einer eindeutigen, globalen kleineren (gmin)-Zahl für das neue Gerät durch die DDI, wobei die Plananforderung die größere Zahl und die locid des neuen Gerätes anzeigt,
in Reaktion auf die Plananforderung: (a) Bestimmen der gmin-Zahl und (b) Liefern der gmin-Zahl zu der DDI durch das DCS, und
wobei die DDI die gmin-Zahl, die durch das DCS geliefert wird und die größere Zahl, dem neuen Gerät zuordnet, so daß auf das neue Gerät in Reaktion auf eine Anforderung zum Öffnen des neuen Gerätes von dem globalen Dateisystem aus zugegriffen werden kann.

8. Verfahren nach Anspruch 7, welches weiterhin die Schritte aufweist:
Ausgeben von Gerätekonfigurationsinformation an die DDI durch den Treiber, einschließlich der Klasseninfor-

mation für das neue Gerät, falls verfügbar, und
Einbeziehen der Klasseninformation, falls verfügbar, in die Plananforderung durch die DDI.

9. Verfahren nach Anspruch 8, welches weiterhin die Schritte aufweist:
nach Empfang der Plananforderung Untersuchen eines lokalen DSO, welches den Geräten zugeordnet ist, deren Klasse dieselbe ist, wie diejenige des neuen Gerätes, durch das DCS, und
durch das DSO Bestimmen der gmin-Zahl, die dem neuen Gerät zugeordnet werden soll.

10. Verfahren nach Anspruch 9, welches weiterhin den Schritt aufweist:
Zugreifen auf das neue Gerät, als ob das neue Gerät zu der dev_enumerate-Klasse gehört, einschließlich Gerät, mit zumindest einmaligem Auftreten, welches durch einen bestimmten Treiber verwaltet wird, wenn die Klasseninformation durch den Gerätetreiber nicht bereitgestellt wird, wobei jeder Fall des Auftretens, welches durch den bestimmten Treiber auf einem bestimmten Knoten verwaltet wird, einzeln numeriert wird.

Es folgen 10 Blatt Zeichnungen

Anhängende Zeichnungen

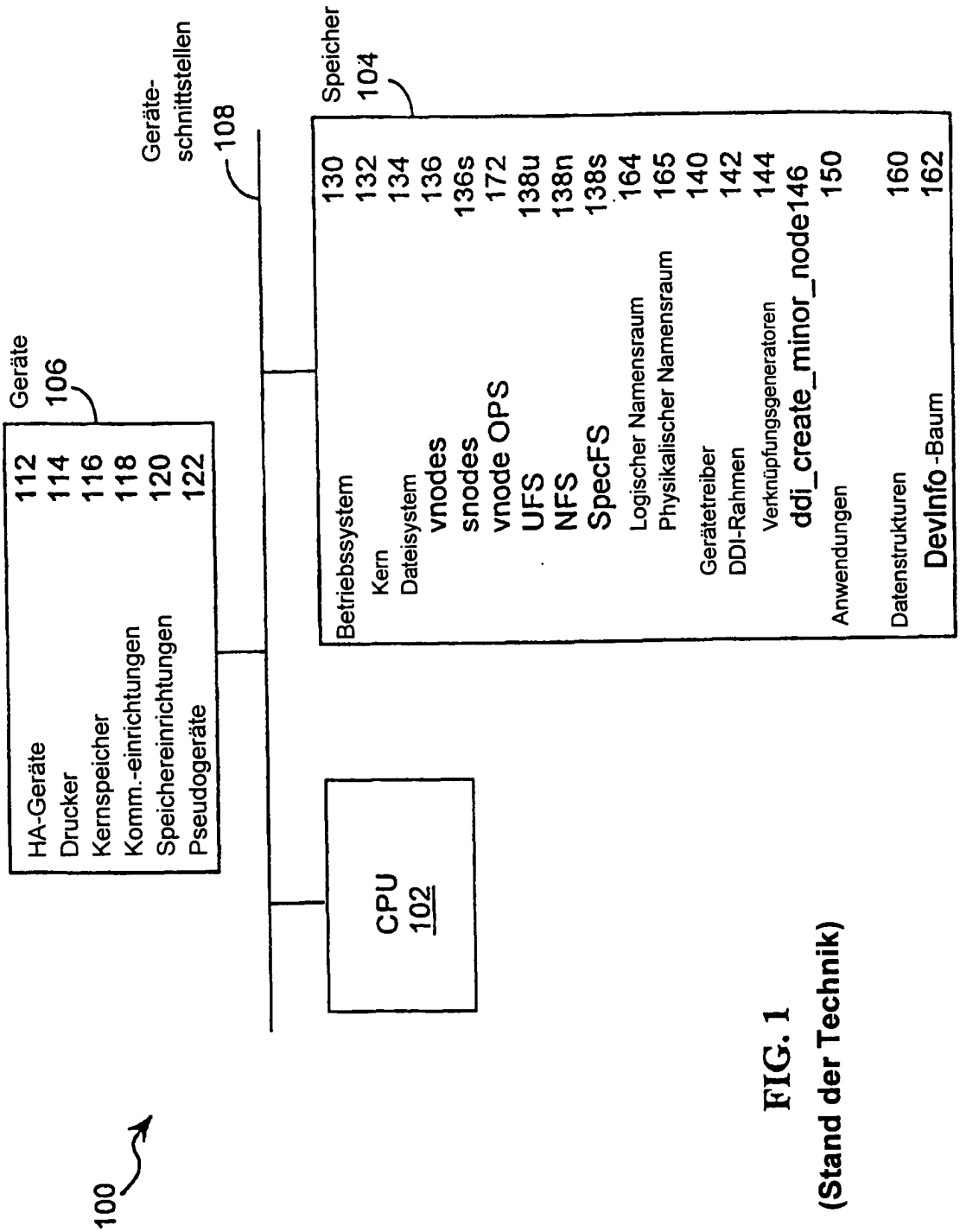

FIG. 1
(Stand der Technik)

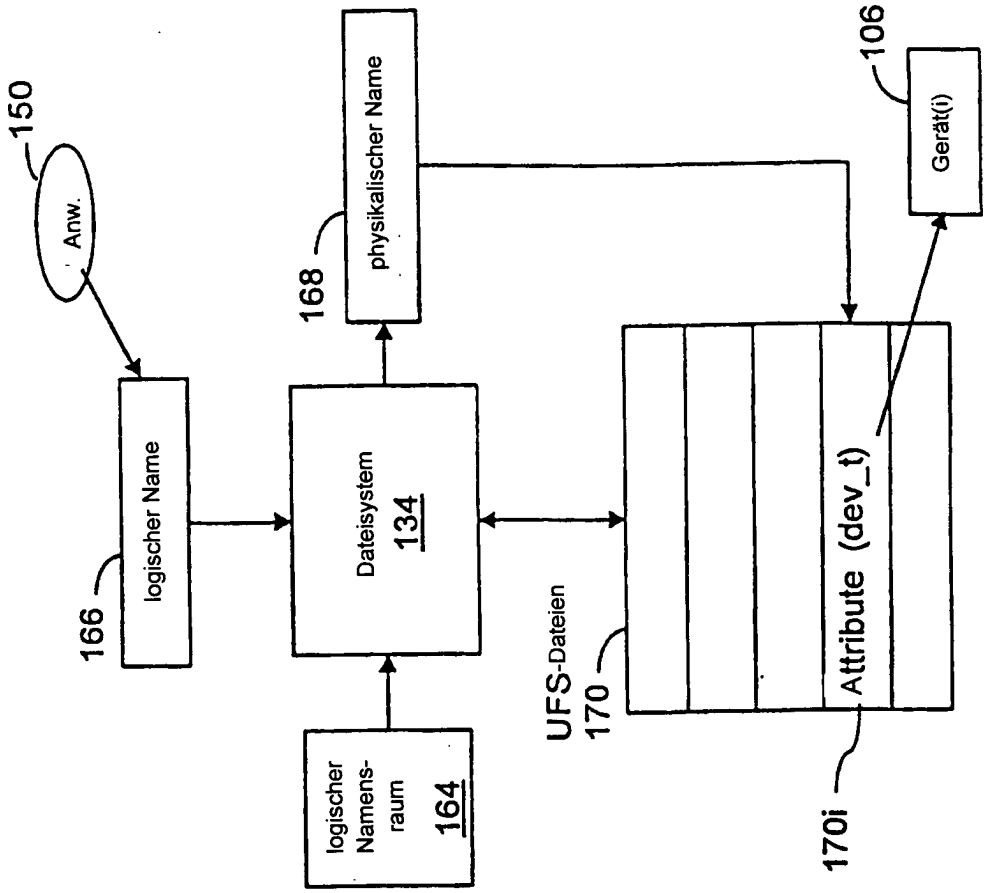

FIG. 2B
(Stand der Technik)

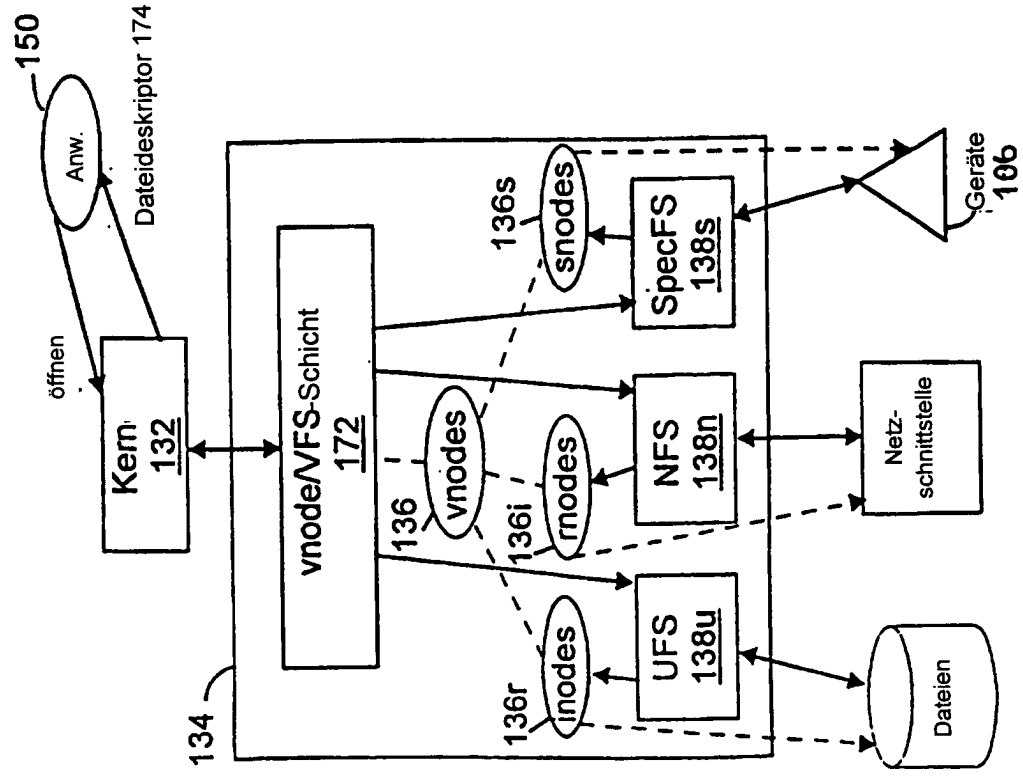

FIG. 2A
(Stand der Technik)

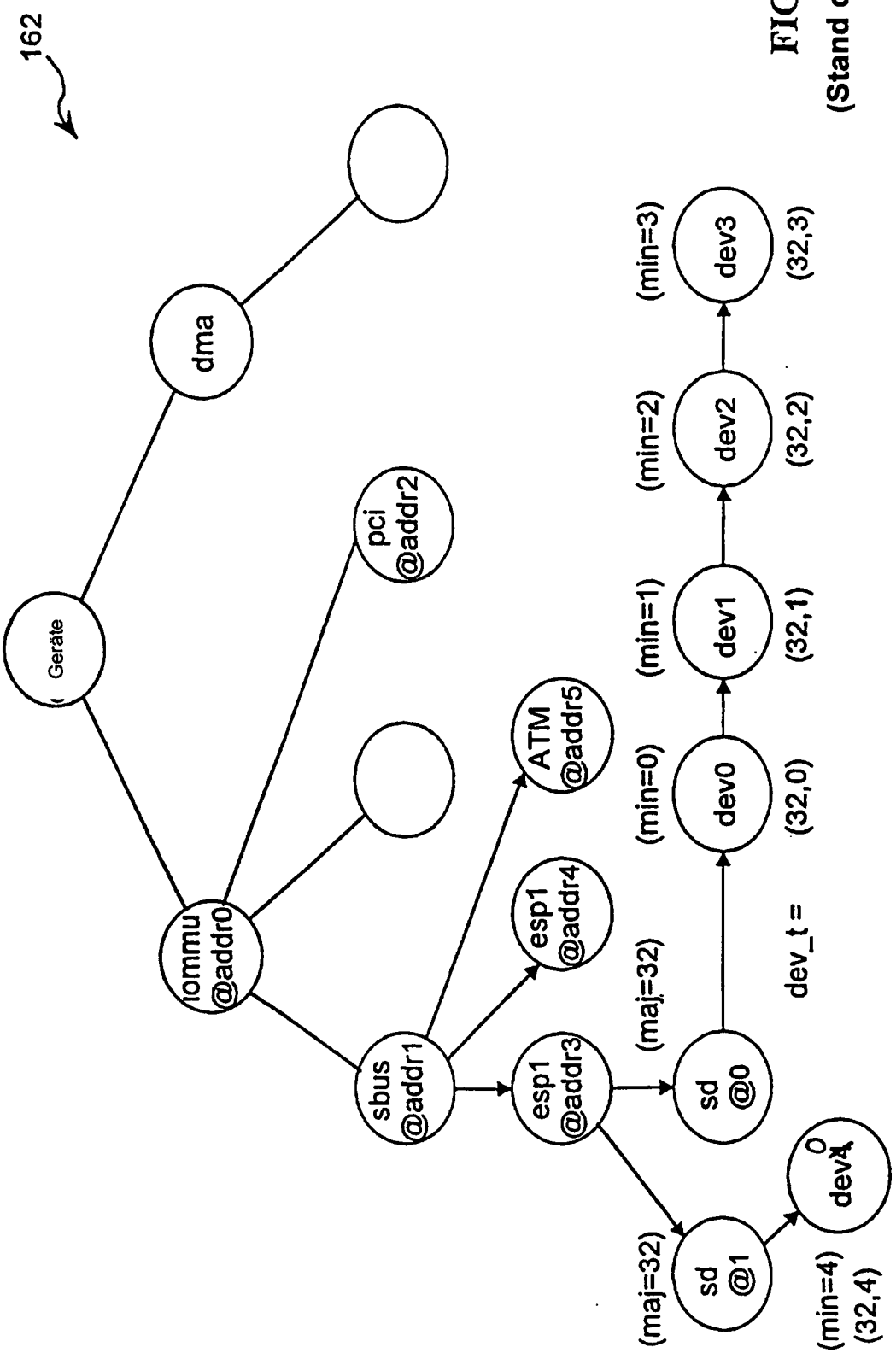

FIG. 3
(Stand der Technik)

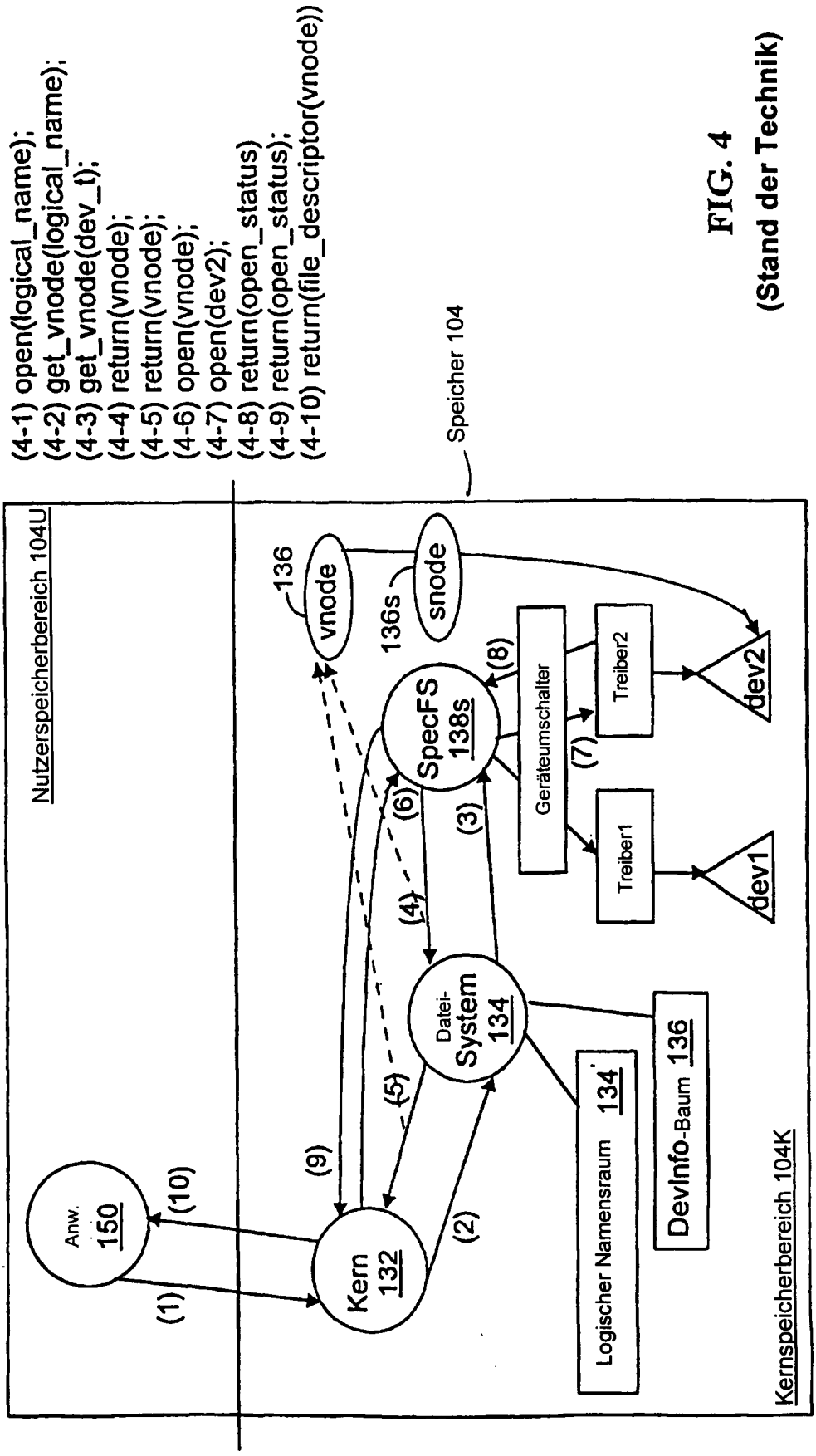

FIG. 4
(Stand der Technik)

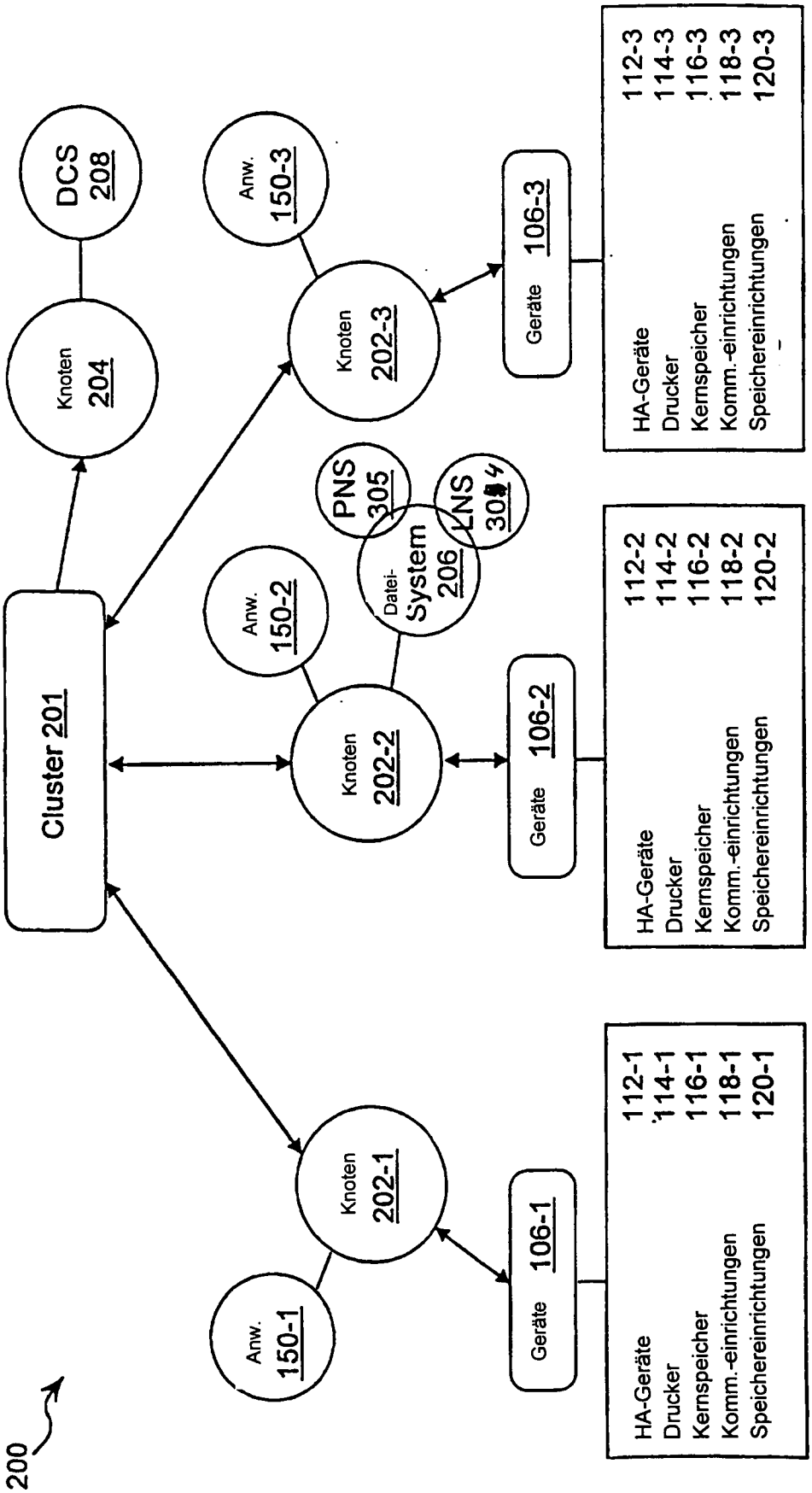
200
Cluster 201
DCS 208
Knoten 204
Anw. 150-1
Anw. 150-2
Anw. 150-3
Knoten 202-1
Knoten 202-2
Knoten 202-3
Geräte 106-1
Geräte 106-2
Geräte 106-3
PNS 305
LNS 304
Datei-System 206
HA-Geräte 112-1
Drucker 114-1
Kernspeicher 116-1
Komm.-einrichtungen 118-1
Speichereinrichtungen 120-1
HA-Geräte 112-2
Drucker 114-2
Kernspeicher 116-2
Komm.-einrichtungen 118-2
Speichereinrichtungen 120-2
HA-Geräte 112-3
Drucker 114-3
Kernspeicher 116-3
Komm.-einrichtungen 118-3
Speichereinrichtungen 120-3

FIG. 5

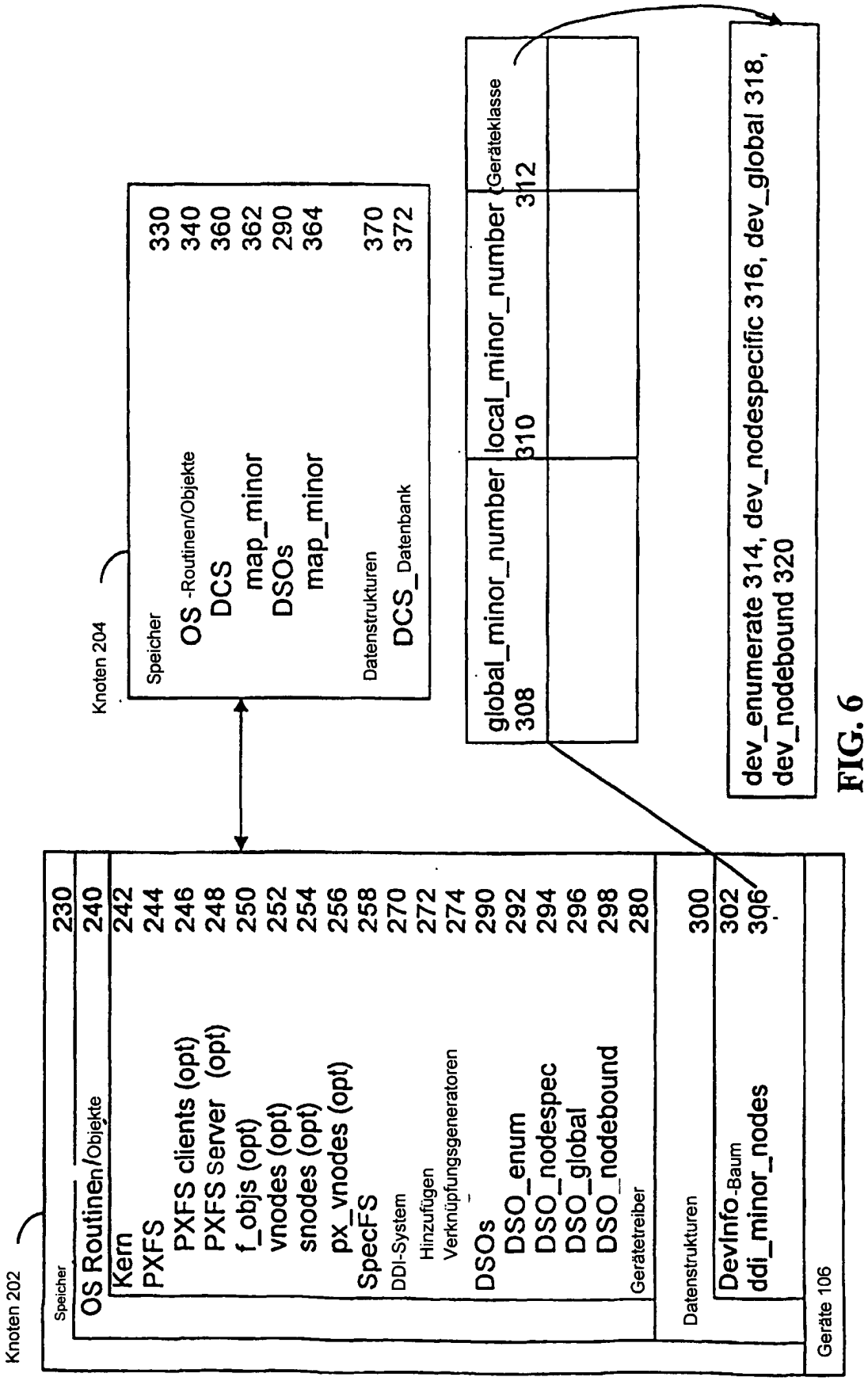
Knoten 202
Speicher
OS Routinen/Objekte 240
Kern 242
PXFS 244
PXFS clients (opt) 246
PXFS Server (opt) 248
f_objs (opt) 250
vnodes (opt) 252
snodes (opt) 254
px_vnodes (opt) 256
SpecFS 258
DDI-System 270
Hinzufügen 272
Verknüpfungsgeneratoren 274
DSOs 290
DSO_enum 292
DSO_nodespec 294
DSO_global 296
DSO_nodebound 298
Gerätetreiber 280
Datenstrukturen 300
DevInfo-Baum 302
ddi_minor_nodes 306
Geräte 106
Knoten 204
Speicher
OS -Routinen/Objekte 330
DCS 340
map_minor 360
DSOs 362
map_minor 290
Datenstrukturen 364
DCS_Datenbank 370
372
global_minor_number 308
local_minor_number 310
Geräteklasse 312
dev_enumerate 314, dev_nodespecific 316, dev_global 318,
dev_nodebound 320

FIG. 6

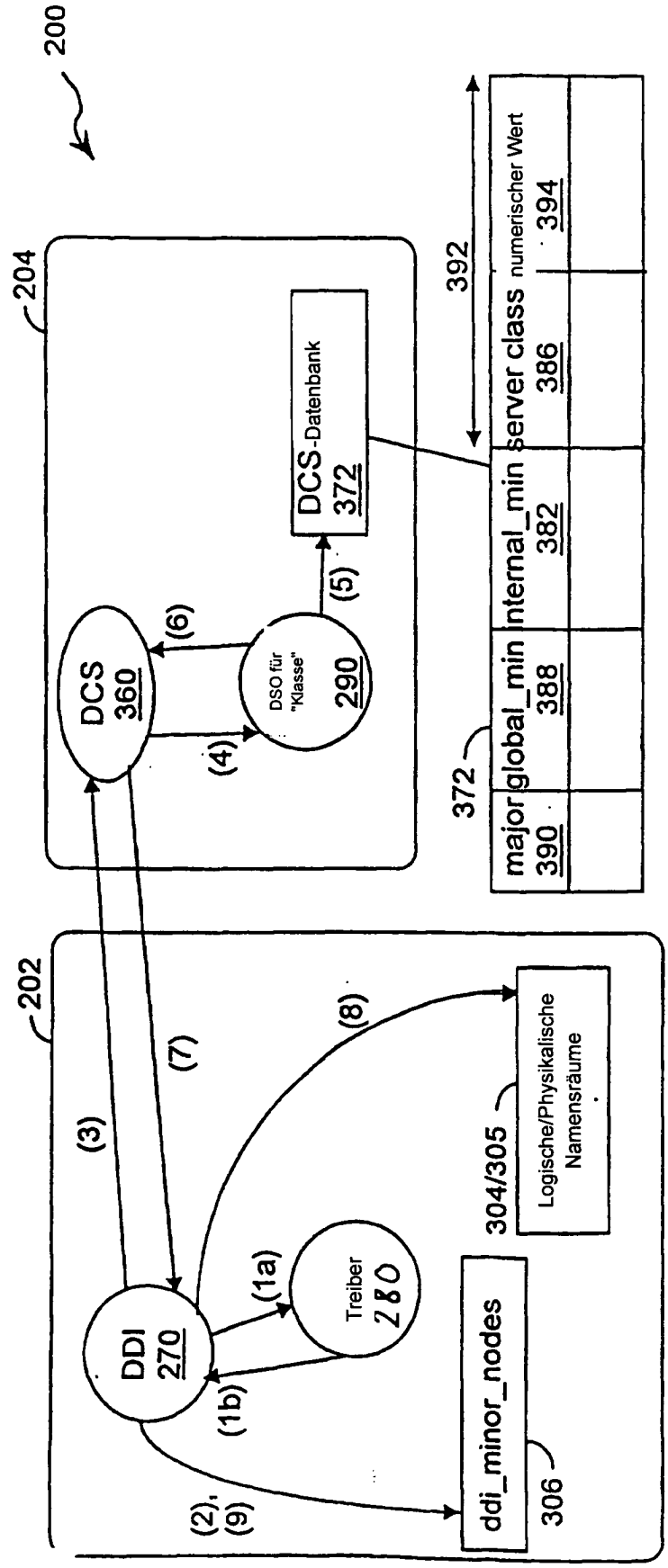

FIG. 7A

(7-1a) attach();
(7-1b) ddi_create_minor_nodes (minor_num 382, minor_name 384, class 386);
(7-2),(7-9) update ddi_minor_nodes (gminor, lminor, class);
(7-3) dc_map_minor (major, lminor, gminor, class);
(7-4) ds_map_minor (major, lminor, gminor);
(7-5) return (gminor);
(7-6) update_DCS_database (major, lminor, gminor, DS_type, DS_num_val);
(7-7) return (gminor);
(7-8) update filesystem;

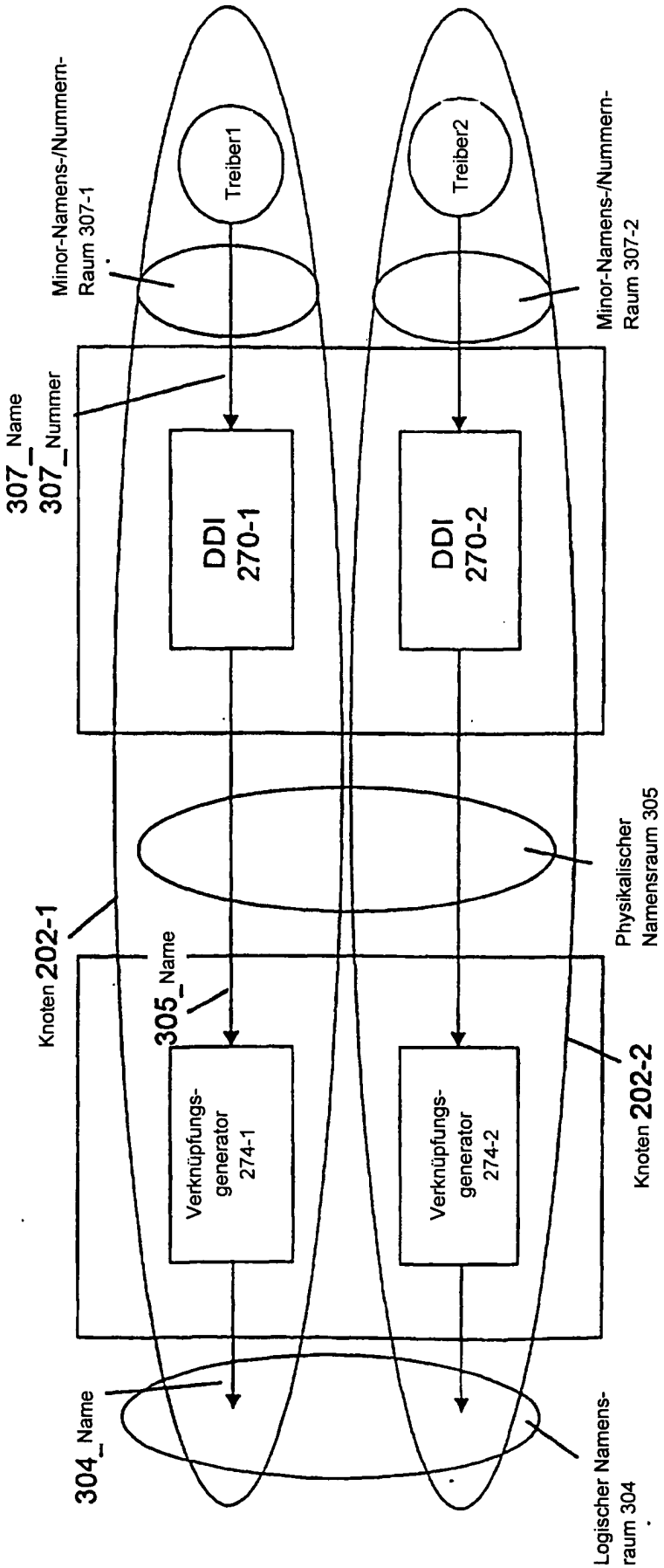
Minor-Namens-/Nummern-Raum 307-1
Treiber1
Treiber2
Minor-Namens-/Nummern-Raum 307-2
307_Name
307_Nummer
DDI 270-1
DDI 270-2
Physikalischer Namensraum 305
Knoten 202-1
305_Name
Verknüpfungs-generator 274-1
Verknüpfungs-generator 274-2
Knoten 202-2
304_Name
Logischer Namens-raum 304

FIG. 7B

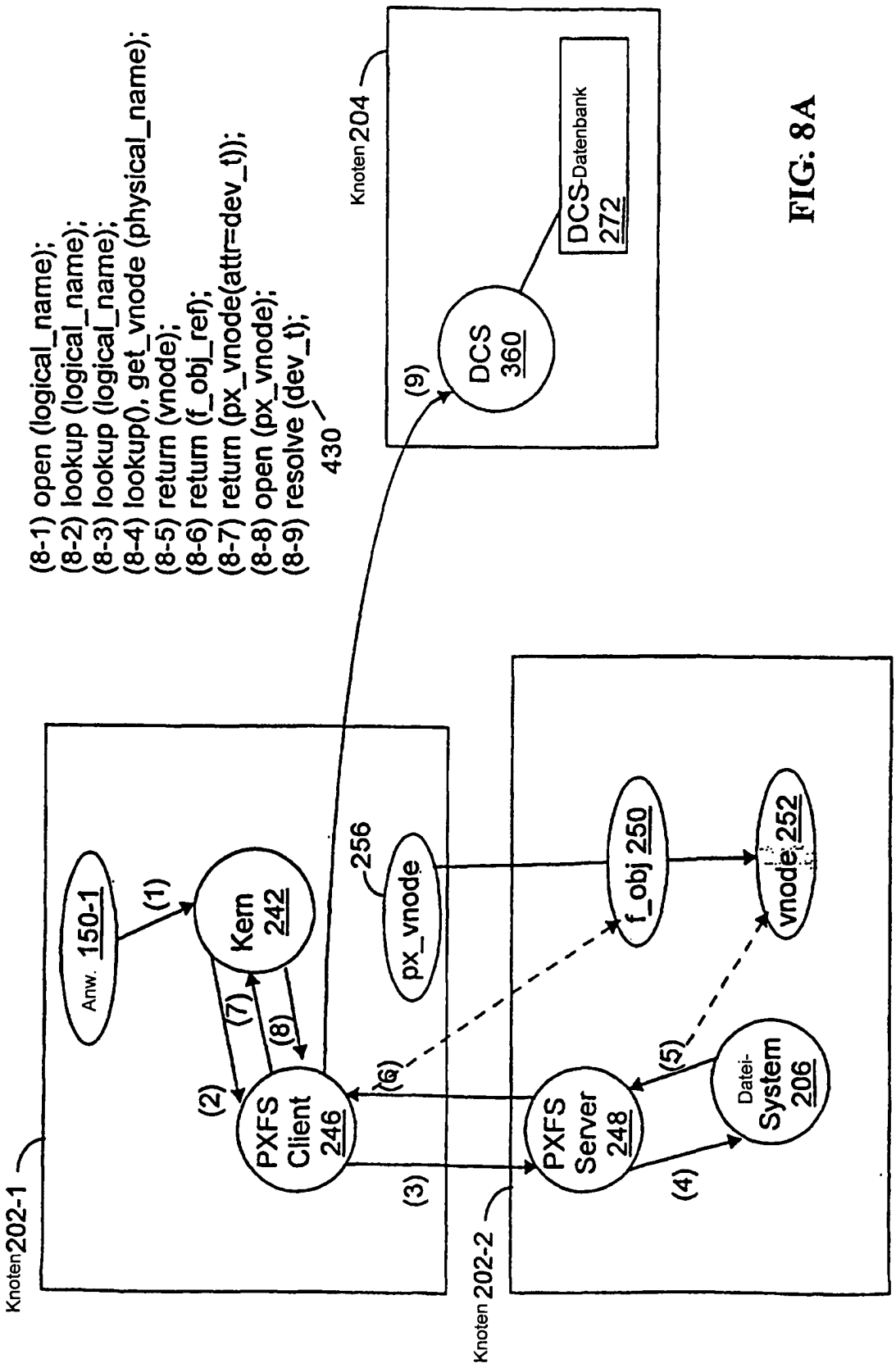
(8-1) open (logical_name);
(8-2) lookup (logical_name);
(8-3) lookup (logical_name);
(8-4) lookup(), get_vnode (physical_name);
(8-5) return (vnode);
(8-6) return (f_obj_ref);
(8-7) return (px_vnode(attr=dev_t));
(8-8) open (px_vnode);
(8-9) resolve (dev_t);
430
Knoten 202-1
Knoten 202-2
Knoten 204
Anw. 150-1
Kern 242
PXFS Client 246
PXFS Server 248
px_vnode
256
f_obj 250
vnode 252
Datei-System 206
DCS 360
DCS-Datenbank 272
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)

FIG. 8A

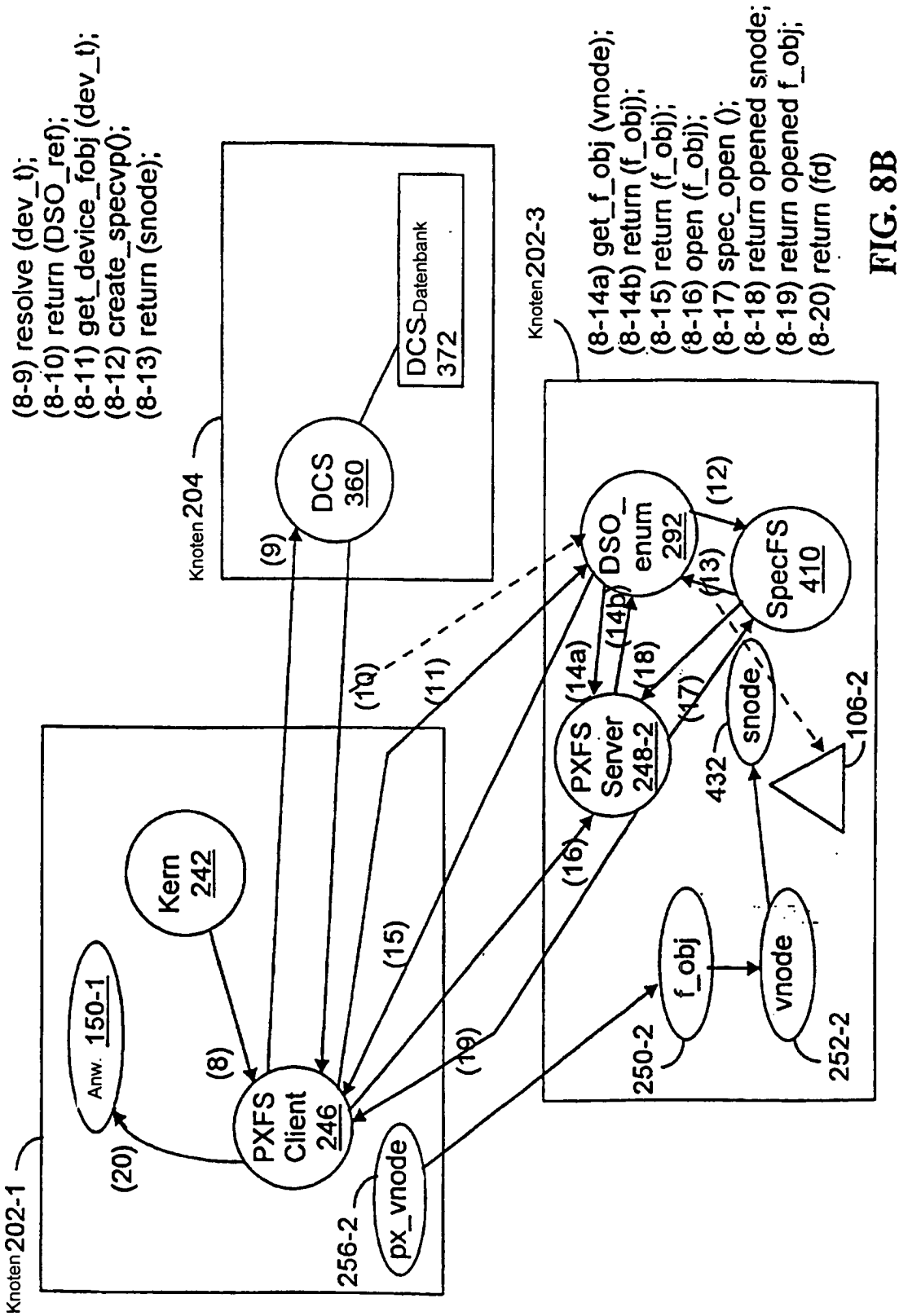
(8-9) resolve (dev_t);
(8-10) return (DSO_ref);
(8-11) get_device_fobj (dev_t);
(8-12) create_specvp();
(8-13) return (snode);
(8-14a) get_f_obj (vnode);
(8-14b) return (f_obj);
(8-15) return (f_obj);
(8-16) open (f_obj);
(8-17) spec_open ();
(8-18) return opened snode;
(8-19) return opened f_obj;
(8-20) return (fd)
Knoten 202-1
Knoten 204
Knoten 202-3
DCS 360
DCS-Datenbank 372
Kern 242
Anw. 150-1
PXFS Client 246
px_vnode
256-2
PXFS Server 248-2
DSO_ enum 292
SpecFS 410
snode
432
106-2
f_obj
250-2
vnode
252-2
(8)
(9)
(10)
(11)
(12)
(13)
(14a)
(14b)
(15)
(16)
(17)
(18)
(19)
(20)

FIG. 8B