Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2667107 A1 2008/05/08

(21) 2 667 107

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(86) Date de dépôt PCT/PCT Filing Date: 2007/10/25
(87) Date publication PCT/PCT Publication Date: 2008/05/08
(85) Entrée phase nationale/National Entry: 2009/04/20
(86) N° demande PCT/PCT Application No.: FR 2007/001765
(87) N° publication PCT/PCT Publication No.: 2008/053098
(30) Priorités/Priorities: 2006/10/26 (FR0609406);
2007/10/24 (FR0707476)

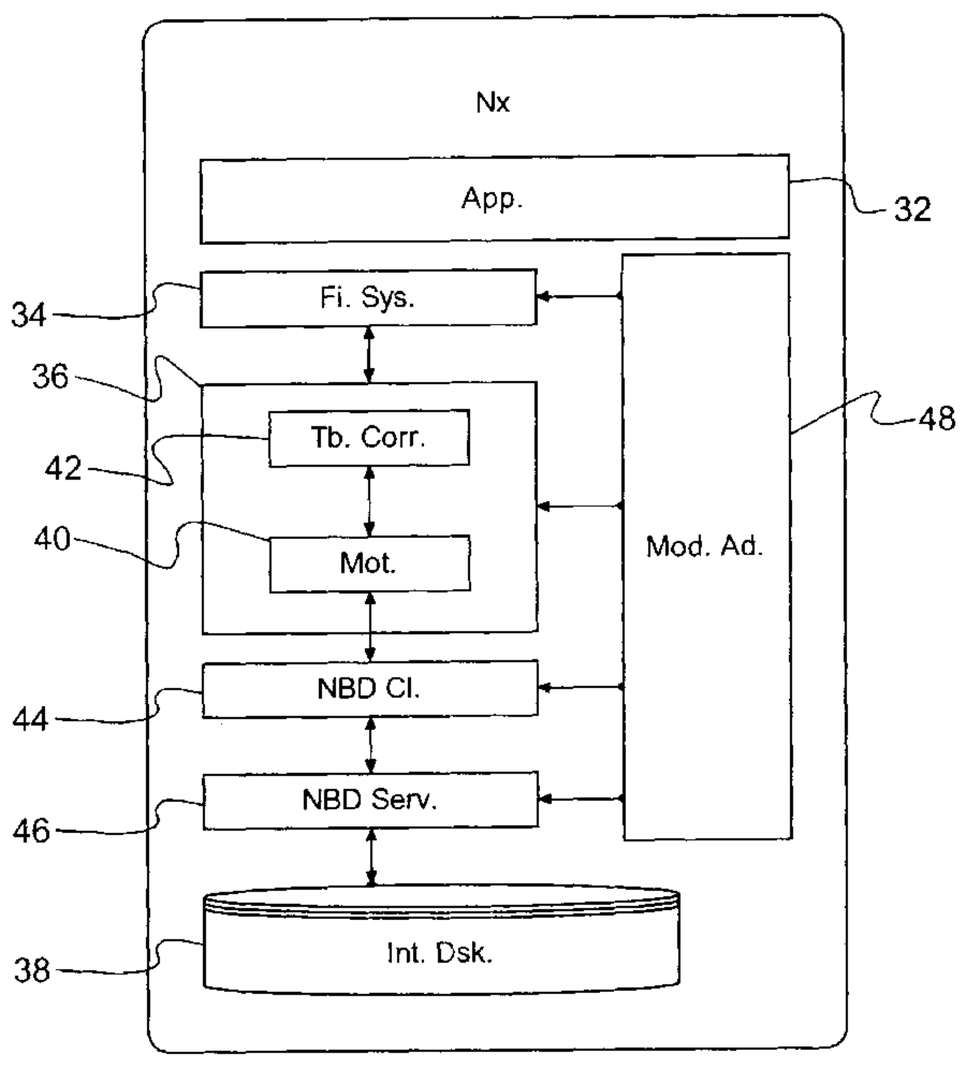
(51) Cl.Int./Int.Cl. *H04L 29/06* (2006.01),
G06F 12/08 (2006.01), *G06F 17/30* (2006.01)

(71) Demandeur/Applicant:
SEANODES, FR

(72) Inventeurs/Inventors:
BEN KHALIFA, NABIL, FR;
COZETTE, OLIVIER, FR;
GUITTENIT, CHRISTOPHE, FR;
RICHARD, SAMUEL, FR

(74) Agent: ROBIC

(54) Titre : SYSTEME INFORMATIQUE AMELIORE COMPRENANT PLUSIEURS NOEUDS EN RESEAU
(54) Title: IMPROVED COMPUTER-BASED SYSTEM COMPRISING SEVERAL NODES IN A NETWORK



(57) Abrégé/Abstract:
Un système informatique comprend plusieurs postes informatiques dits noeuds interconnectés en un réseau, des nœuds stockants comprennent une unité de mémoire locale (38) à accès direct, et un serveur de stockage (46) pour gérer l'accès à cette unité sur la base d'une file d'attente, des nœuds applicatifs, comprennent chacun : un environnement d'application avec une représentation de fichiers accessibles sous forme d'adresses de blocs désignant une adresse physique sur une unité de mémoire locale (33), et un client de stockage (44) capable d'interagir avec un serveur de stockage (46), sur la base d'une requête d'accès désignant une adresse de blocs. Un serveur de stockage (46) comporte un ordonnanceur capable d'exécuter les requêtes d'accès contenues dans sa file d'attente (64) dans un ordre déterminé. Cet ordre est déterminé en fonction d'un jeu de règles formant critères de performance, et impliquant un ou plusieurs paramètres d'état de la file d'attente et/ou du nœud stockant sur lequel il réside.

(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

PCT

(43) Date de la publication internationale
8 mai 2008 (08.05.2008)

(10) Numéro de publication internationale
WO 2008/053098 A3

(51) Classification internationale des brevets :
H04L 29/06 (2006.01) *G06F 17/30* (2006.01)
G06F 12/08 (2006.01)

(21) Numéro de la demande internationale :
PCT/FR2007/001765

(22) Date de dépôt international :
25 octobre 2007 (25.10.2007)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :

0609406	26 octobre 2006 (26.10.2006)	FR
0707476	24 octobre 2007 (24.10.2007)	FR

(71) Déposant *(pour tous les États désignés sauf US)* : **SEANODES** [FR/FR]; 25-27 boulevard Victor Hugo, Sciences Parc du Perget, Batiment Platon, F-31770 Colomiers (FR).

(72) Inventeurs; et

(75) Inventeurs/Déposants *(pour US seulement)* : **BEN KHALIFA, Nabil** [TN/FR]; 7, Boulevard Monplaisir, F-31400 Toulouse (FR). **COZETTE, Olivier** [FR/FR]; 18, rue Pénent, F-31100 Toulouse (FR). **GUITTENIT, Christophe** [FR/FR]; 2, rue Ninau, F-31000 Toulouse (FR). **RICHARD, Samuel** [FR/FR]; Batiment D1, 12, rue Louis Vitet, F-31400 Toulouse (FR).

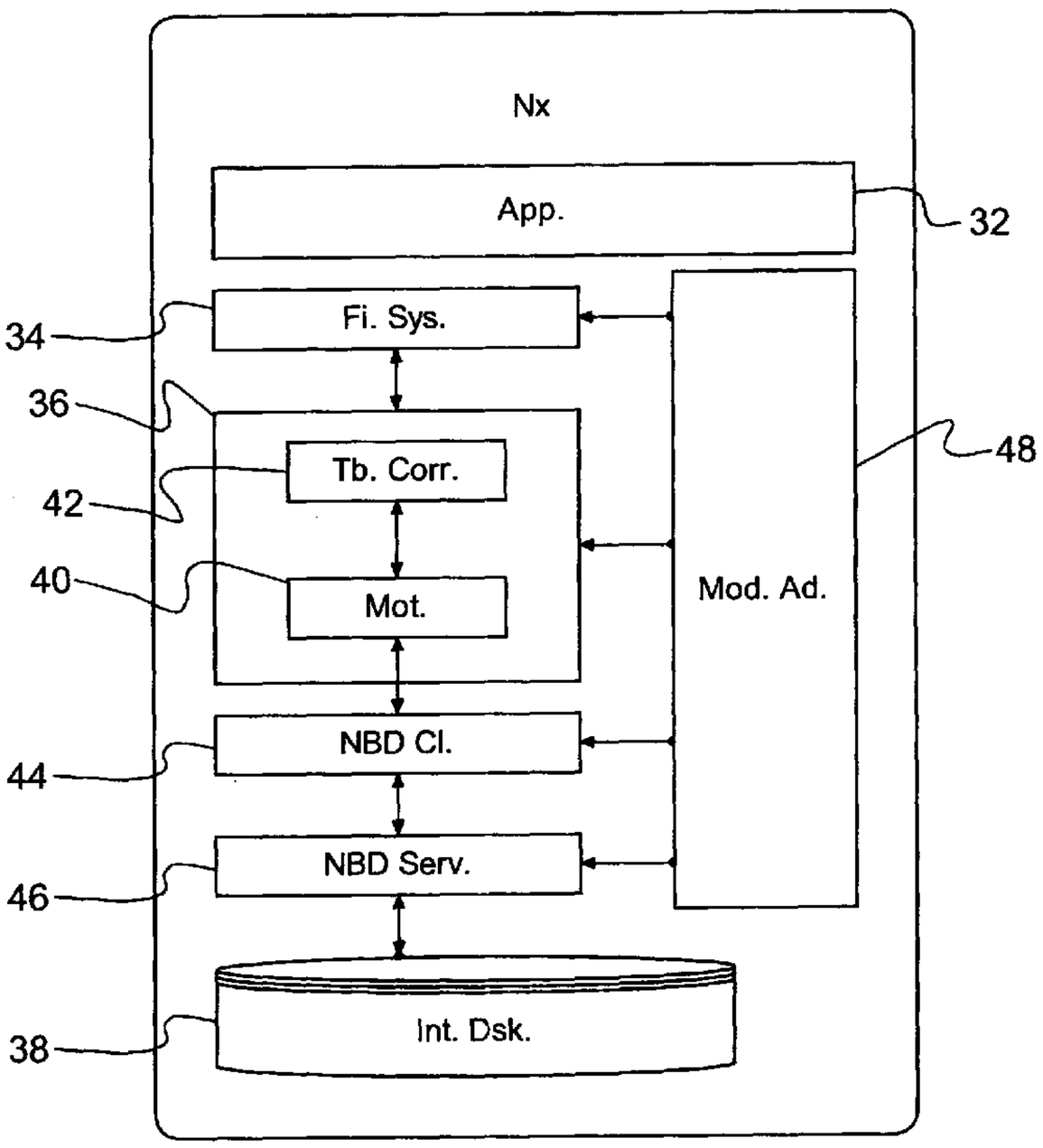
(74) Mandataire : **BEZAULT, Jean**; Cabinet Netter, 36, avenue Hoche, F-75008 Paris (FR).

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR,

[Suite sur la page suivante]

(54) Title: IMPROVED COMPUTER-BASED SYSTEM COMPRISING SEVERAL NODES IN A NETWORK

(54) Titre : SYSTEME INFORMATIQUE AMELIORE COMPRENANT PLUSIEURS NOEUDS EN RESEAU



(57) Abstract: A computer-based system comprises several computer-based facilities termed nodes interconnected in a network, - storing nodes comprise a direct-access local memory unit (38), and a storage server (46) for managing access to this unit on the basis of a queue, - application nodes, each comprising: * an application environment with a representation of files that are accessible in the form of addresses of blocks designating a physical address on a local memory unit (33), and * a storage client (44) capable of interacting with a storage server (46), on the basis of an access request designating a block address. A storage server (46) comprises a scheduler capable of executing the access requests contained in its queue (64) in a determined order. This order is determined as a function of a set of rules forming performance criteria, and involving one or more state parameters of the queue and/or of the storing node on which it resides.

[Suite sur la page suivante]

WO 2008/053098 A3

WO 2008/053098 A3

LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* **:** ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *avec rapport de recherche internationale*

— *avant l'expiration du délai prévu pour la modification des revendications, sera republiée si des modifications sont reçues*

(88) Date de publication du rapport de recherche internationale: 9 octobre 2008

(57) Abrégé : Un système informatique comprend plusieurs postes informatiques dits noeuds interconnectés en un réseau, des nœuds stockants comprennent une unité de mémoire locale (38) à accès direct, et un serveur de stockage (46) pour gérer l'accès à cette unité sur la base d'une file d'attente, des nœuds applicatifs, comprennent chacun : un environnement d'application avec une représentation de fichiers accessibles sous forme d'adresses de blocs désignant une adresse physique sur une unité de mémoire locale (33), et un client de stockage (44) capable d'interagir avec un serveur de stockage (46), sur la base d'une requête d'accès désignant une adresse de blocs. Un serveur de stockage (46) comporte un ordonnanceur capable d'exécuter les requêtes d'accès contenues dans sa file d'attente (64) dans un ordre déterminé. Cet ordre est déterminé en fonction d'un jeu de règles formant critères de performance, et impliquant un ou plusieurs paramètres d'état de la file d'attente et/ou du nœud stockant sur lequel il réside.

Système informatique amélioré comprenant plusieurs noeuds en réseau

L'invention concerne les systèmes informatiques comprenant plusieurs postes informatiques dits noeuds interconnectés en un réseau.

Les réseaux modernes comportent des stations utilisateurs qui sont reliées à un ou plusieurs serveurs et peuvent partager des applications et/ou des espaces de stockage de manière locale ou distante.

Dans le cadre d'applications partagées faisant usage d'une quantité importante de données ou dans le cadre du partage d'une quantité importante de données, il est fréquent de faire appel à des systèmes de stockage spécialisés, tels que les Storage Area Network (ou SAN).

L'utilisation de ces systèmes perfectionnés présente certains désavantages, comme les coûts associés, les limitations de performance et d'extensibilité, et la lourdeur générale de l'installation qui leur correspond.

Par ailleurs, avec les réseaux modernes, l'utilisation de ces systèmes perfectionnés représente une sous utilisation du matériel déjà présent dans le réseau.

L'invention vient améliorer la situation.

À cet effet, l'invention propose un système informatique comprenant plusieurs postes informatiques dits noeuds interconnectés en un réseau. Certains au moins des nœuds, dits stockants, comprennent au moins une unité de mémoire locale à accès direct, et un serveur de stockage agencé pour gérer l'accès à cette unité de mémoire locale sur la base d'une file d'attente de requêtes d'accès.

Certains au moins des nœuds, dits applicatifs, comprennent chacun :

* un environnement d'application muni :

- d'un gestionnaire de système de fichiers, agencé pour maintenir une représentation de fichiers accessibles depuis ce nœud sous forme d'adresses virtuelles, et

- d'un module de correspondance, capable de maintenir une correspondance entre chaque adresse virtuelle et au moins une adresse physique sur une unité de mémoire locale d'un noeud stockant, et

* un client de stockage capable d'interagir avec l'un quelconque des serveurs de stockage, sur la base d'une requête d'accès désignant une adresse physique.

Dans ce système informatique, au moins l'un des serveurs de stockage comporte un ordonnanceur capable d'exécuter les requêtes d'accès contenues dans sa file d'attente dans un ordre déterminé, et en ce que l'ordonnanceur est agencé pour déterminer cet ordre en fonction d'un jeu de règles formant critères de performance, et impliquant un ou plusieurs paramètres d'état de la file d'attente et/ou du nœud stockant sur lequel il réside.

Un tel système informatique présente l'avantage d'utiliser les ressources de stockage intrinsèques des stations (qui seront également appelées nœuds) du réseau afin de stocker de manière efficace les données. Dans ces stations, le serveur de stockage peut être utilisé pour optimiser l'utilisation du réseau et des ressources de stockage. Ce système permet ainsi de faire un usage maximal des capacités intrinsèques du réseau sans faire usage de systèmes spécialisés.

L'invention concerne également un procédé de gestion de données, applicable dans un réseau comprenant plusieurs postes informatiques interconnectés, dits noeuds, comportant les étapes suivantes :

a. émettre une requête de fichier depuis un nœud applicatif du réseau, sur la base d'une représentation sous forme d'adresses virtuelles,

b. sur la base d'une correspondance entre chaque adresse virtuelle et au moins une adresse physique sur une unité de mémoire locale d'un noeud stockant du réseau, déterminer au moins une adresse physique correspondant à ladite adresse virtuelle,

c. émettre une requête d'accès désignant ladite adresse physique vers un serveur de stockage gérant l'accès à son unité de mémoire locale, et

d. placer la requête d'accès dans une file d'attente dudit serveur de stockage, et exécuter les requêtes d'accès contenues dans ladite file d'attente, dans un ordre déterminé en fonction de d'un jeu de règles formant critères de performance, et impliquant un ou plusieurs paramètres d'état de la ou des unités de mémoire locale et/ou du nœud stockant concerné (charge processeur, occupation de la mémoire centrale, etc..).

D'autres avantages et caractéristiques de l'invention apparaîtront mieux à la lecture de la description qui suit d'exemples, donnée à titre illustratif et non limitatif, à partir des dessins sur lesquels :

- la figure 1 montre une vue fonctionnelle générale d'un système informatique selon l'invention,

- la figure 2 montre un exemple d'implémentation logique du système de la figure 1,

- la figure 3 montre un exemple de composition d'un élément de la figure 2,

- la figure 4 montre un procédé d'accès à un fichier dans le système de la figure 1,

- la figure 5 montre un exemple d'implémentation fonctionnelle d'une partie du procédé de la figure 4,

- la figure 6 montre un exemple d'une boucle d'ordonnancement et d'exécution dans un serveur de stockage dans le cadre de l'implémentation de la figure 5,

- les figures 7 à 10 montrent des exemples de fonctions de la figure 6,

- la figure 11 montre un exemple d'une boucle d'ordonnancement et d'exécution dans un serveur de stockage en variante,

- la figure 12 montre un exemple d'une fonction de la figure 11, et

- les figures 13 à 15 montrent des exemples de fonctions de la figure 12.

Les dessins et la description ci-après contiennent, pour l'essentiel, des éléments de caractère certain. Ils pourront donc non seulement servir à mieux faire comprendre la présente invention, mais aussi contribuer à sa définition, le cas échéant.

La figure 1 représente un schéma général d'un système informatique selon l'invention. Dans ce système, un environnement d'application 2 a accès à un gestionnaire de système de fichiers 4. Une couche de virtualisation 6 établit la correspondance entre le gestionnaire de système de fichiers 4 et des serveurs de stockage 8.

La figure 2 représente une implémentation logique du système de la figure 1. Dans cette implémentation, un ensemble de stations 10, également appelées ici noeuds sont interconnectées en un réseau dont elles constituent les ressources physiques et applicatives.

Dans l'exemple ici décrit, le réseau est constitué de 5 stations, notées Ni avec i variant entre 1 et 5. L'environnement d'application 2 est réalisé en une couche applicative répartie 12 sur les N1, N2 et N3, en une couche applicative 14 sur le N4 et une couche applicative 16 sur le N5.

On notera que le terme poste ou station utilisé ici doit être interprété de manière générale, et comme désignant des éléments informatiques du réseau sur lesquels tournent des applications ou des programmes de serveur, ou les deux.
Le gestionnaire de système de fichiers 4 est réalisé en un système de fichiers réparti 18, et deux systèmes de fichiers non répartis 20 et 22. Le système 18 est réparti sur les N1, N2 et N3 et définit l'ensemble des fichiers accessibles depuis la couche applicative répartie 12. Les systèmes de fichiers 20 et 22 définissent respectivement l'ensemble des fichiers accessibles depuis les couches applicatives 14 et 16.

Les fichiers désignés par les systèmes de fichiers 18, 20 et 22 sont stockés physiquement dans un espace de stockage virtuel 24 qui est réparti sur l'ensemble des Ni avec i variant entre 1 et 5. L'espace de stockage virtuel 24 est ici réparti en un espace logique partagé 26, et deux espaces logiques privés 28 et 30.

L'espace logique partagé 26 correspond à l'espace accessible depuis la couche applicative répartie 12 au moyen du système de fichiers réparti 18, et les espaces logiques privés 28 et 30 à l'espace accessible depuis les couches applicatives 14 et 16 au moyen des systèmes de fichiers 20 et 22.

L'espace logique 26 est réparti sur les N1, N2 et N3, l'espace logique privé 28 sur les N3 et N4, et l'espace logique privé 30 sur le N5.

Ainsi, une application de la couche 12 (respectivement 14, 16) "voit" les données stockées dans l'espace logique 26 (respectivement 28, 30) au moyen du système de fichiers 18 (respectivement 20, 22), bien que celles-ci ne soient pas forcément physiquement présentes sur l'un des disques de stockage de la station 10 qui utilise cette application.

Par ailleurs, les espaces 26, 28 et 30 sont purement logiques, c'est-à-dire qu'ils ne représentent pas directement des espaces de stockage physiques. Les espaces logiques sont cartographiés au moyen d'adresses virtuelles qui sont référencées ou contenues dans les systèmes de fichiers 18, 20 et 22.

Pour accéder aux données de ces fichiers, il est nécessaire de faire appel à un module de correspondance. Le module de correspondance contient une table de correspondance entre les adresses virtuelles des données dans les espaces logiques et des adresses physiques qui désignent les espaces de stockage physiques dans lesquels ces données sont réellement stockées.

Plusieurs réalisations sont possibles pour le module de correspondance. La répartition des espaces de stockage physiques décrite ici est un exemple destiné à montrer la portée très générale de l'invention.

Comme on peut le voir dans l'exemple présenté, chaque station est utilisée à la fois pour la couche applicative et pour la couche de stockage. Cette multifonctionnalité permet d'utiliser l'espace libre sur l'ensemble des stations du réseau, plutôt que laisser cet espace inoccupé.

Dans le cadre de l'invention, il serait cependant possible de spécialiser certaines des stations, et de créer un nœud dédié au stockage ou un noeud dédié à des applications.

Cela signifie que, dans le cadre de l'invention, toute station peut jouer un rôle de nœud applicatif, un rôle de nœud stockant, ou encore ces deux rôles à la fois.

L'ensemble des ressources applicatives, de stockage et de système de fichiers peuvent être intégrées localement sur chaque station, ou bien réparties sur les stations du réseau.

C'est par exemple le cas des stations N1, N2 et N3, dont les ressources sont intégralement réparties, tant au niveau applicatif qu'au niveau du système de fichiers et du stockage.

La figure 3 représente un exemple d'architecture d'une station 10 de la figure 2. La station représentée dans cet exemple peut représenter l'une des stations N1, N2 ou N3.

La station Nx présente individuellement une structure semblable à celle de la structure globale représentée sur la figure 1. Elle comporte ainsi une couche applicative 32, un système de fichiers 34, une couche de virtualisation 36 et un espace de stockage 38 sous la forme d'une mémoire locale à accès direct.

La couche de virtualisation 36 comporte un moteur 40 et une table de correspondance 42. L'accès direct à l'espace de stockage 38 est géré par un client de stockage 44 et un serveur de stockage 46. Les rôles et les fonctionnements de ces éléments seront précisés plus bas.

L'exemple décrit ici représente un mode de réalisation perfectionné de l'invention, dans lequel toutes les ressources, tant applicatives que de stockage, sont réparties sur le réseau.

Cela signifie par exemple que le système de fichiers 34 n'est pas intégralement présent sur cette station, mais réparti sur plusieurs d'entre elles, et que l'accès

à celui-ci implique la communication avec d'autres nœuds du réseau qui contiennent les données recherchées.

Il en est de même pour la couche de virtualisation 36, le client de stockage 44 et le serveur de stockage 46. La répartition de ces éléments est gérée au moyen d'un module d'administration 48.

Le module d'administration 48 est principalement utilisé lors de la création et de la mise à jour des espaces logiques. Lors de la création ou de la modification d'un espace logique, le module d'administration 48 appelle la couche de virtualisation 36 pour créer la table de correspondance entre chaque adresse virtuelle de l'espace logique et une adresse physique sur un nœud de stockage donné.

Ensuite, les correspondances entre un fichier accessible par ce système de fichiers et les adresses virtuelles des données qui composent ce fichier sont réalisée au niveau du système de fichiers qui exploite cet espace logique, les données "physiques" étant stockées dans les adresses physiques associées dans la table de correspondance aux adresses virtuelles, conformément à la cartographie établie lors de la création de l'espace logique.

Cela signifie que, dès la création d'un espace logique par le module d'administration, les correspondances entre les adresses virtuelles et les adresses physiques sont établies. Les adresses virtuelles apparaissent ainsi "vides" au système de fichier accédant à l'espace logique, bien que les adresses physiques qui leur correspondent soient déjà "réservées" par le biais de la table de correspondance.

C'est lorsque le lien entre les données des fichiers de cet espace et les adresses virtuelles de ces données est établi que les adresses physiques sont remplies.

Le travail accompli par la couche de virtualisation peut être réalisé de différentes manières. Dans un exemple de réalisation, la couche de virtualisation distribue les données sur les ressources de stockage hétérogènes pour trouver le meilleur compromis entre l'exploitation du débit des ressources de stockage du réseau, et l'exploitation de capacité de stockage de ces ressources. Un exemple de cette couche de virtualisation est décrit dans les paragraphes [0046] à [0062] du brevet EP 1 454 269 B1.

La couche de virtualisation 36 peut également incorporer un mécanisme de sauvegarde des données écrites. Ce mécanisme peut par exemple reposer sur une duplication sélective de chaque requête en écriture avec des adresses physiques situées sur des espaces de stockages physiques situés sur des stations distinctes, à la manière d'un RAID.

La table de correspondance 42 n'est pas nécessairement un simple tableau. Elle peut notamment contenir des informations de configuration concernant le ou les espaces logiques dont elle maintient les correspondances. Dans ce cas, elle peut notamment interagir avec des mécanismes de la couche de virtualisation 36 qui pour mettre à jour la répartition des correspondances adresses virtuelles / adresses physiques afin d'assurer le meilleur compromis entre l'exploitation du débit des ressources de stockage du réseau, et l'exploitation de capacité de stockage de ces ressources. Un exemple de réalisation de ces mécanismes est décrit dans les paragraphes [0063] à [0075] du brevet EP 1 454 269 B1.

Pour la suite de la description, il importe peu que les ressources considérées soient réparties ou pas.

Afin de mieux comprendre l'invention, il convient de bien différencier la couche applicative de la couche de stockage. En effet, la gestion de l'accès aux données stockées dans la couche de stockage est une approche qui présente de nombreux avantages par rapport à l'existant.

La figure 4 représente un procédé mis en œuvre par le système pour accéder à un fichier.

L'accès à un fichier par une application de la couche applicative d'un nœud donné est initialisé par une requête d'accès fichier 50. La requête d'accès fichier 50 comporte :

- un identifiant du fichier concerné pour le système de fichiers et une adresse dans ce fichier,

- la taille de la requête, c'est-à-dire le nombre de bits à accéder à la suite de l'adresse du fichier visé, et

- le type de requête, à savoir la lecture ou l'écriture.

Dans une étape 52, le système de fichiers détermine une ou plusieurs adresses virtuelles pour les données de ce fichier, et génère une ou plusieurs requêtes d'accès virtuel sur la base de la requête 50 et de ces adresses virtuelles.

Les requêtes d'accès virtuel comportent chacune :

- l'adresse virtuelle visée,

- la taille de la requête, c'est-à-dire le nombre de bits à accéder à la suite de l'adresse virtuelle visée, et

- le type de requête, qui est identique à celui de la requête 50.

Si l'on se rapporte au système décrit sur la figure 2, l'étape 52 consiste à déterminer l'espace logique et la ou les adresses virtuelles sur cet espace désignées par la requête 50, et à produire une ou plusieurs requêtes "virtuelles".

Il existe une différence de niveau entre les requêtes d'accès fichiers et les requêtes d'accès virtuel. En effet, une requête d'accès fichier va viser le contenu d'une quantité importante d'adresses virtuelles, afin de permettre de reconstituer le contenu d'un fichier, alors qu'une requête virtuelle vise le contenu d'un bloc de données associé à cette adresse.

La ou les requêtes d'accès virtuel obtenues sont alors transmises à la couche de virtualisation, qui détermine la ou les adresses physiques et les espaces de stockage correspondants dans une étape 54.

Pour déterminer les adresses physiques, la couche de virtualisation opère en utilisant le moteur 40 et la table de correspondance 42.

Dans le cadre d'une requête d'accès en lecture, le fichier recherché existe déjà dans un espace de stockage 38, et le moteur 40 appelle la table de correspondance 42 avec la ou les adresses virtuelles pour déterminer par correspondance la ou les adresses physiques des données du fichier.

Dans le cadre d'une requête d'accès en écriture, le fichier n'existe pas forcément de manière préalable dans un espace de stockage 38. Néanmoins, comme on l'a vu plus haut, les correspondances entre adresses virtuelles et adresses physiques sont figées, et le moteur 40 opère donc de la même manière que dans le cadre d'une requête en lecture pour déterminer la ou les adresses physiques des données.

Dans tous les cas, une fois que le moteur 40 a déterminé les adresses physiques, il génère dans une étape 56 des requêtes d'accès physique qu'il transmet au client de stockage 44.

Dans l'étape 56, les requêtes d'accès physique sont générées sur la base de la requête 50 et de la ou des adresses physiques déterminées à l'étape 54.

Ces requêtes comportent :

- l'adresse physique visée ;

- la taille de la requête, c'est-à-dire le nombre de bits à accéder à la suite de l'adresse physique visée par la requête ; et

- le type d'action visée, à savoir la lecture ou l'écriture.

L'adresse physique et la taille de la requête sont obtenues directement de l'étape 54, et le type de la requête est hérité du type de la requête d'accès virtuel concernée.

Une boucle est alors lancée, dans laquelle une condition d'arrêt 58 est atteinte lorsqu'une requête d'accès physique à été émise au client de stockage 44 pour toutes les adresses physiques obtenues à l'étape 52.

En fait, chaque requête d'accès physique est placée dans une file d'attente de requête du client de stockage 44 pour exécution dans une étape 60. Le client de stockage 44 peut optionnellement comporter plusieurs files d'attente, par exemple une file d'attente par serveur de stockage 46 avec lequel il interagit.

Dans cette boucle, toutes les requêtes d'accès physique de l'étape 56 sont représentées comme exécutées successivement pour des raisons de simplicité. Cependant, l'exécution peut également être réalisée en parallèle, et pas seulement en série.

Dans l'exemple décrit, des requêtes sont transmises de couche en couche, jusqu'à la couche d'accès physique. Il serait cependant possible de déterminer et de transmettre uniquement des adresses (virtuelles puis physiques), et de récupérer, au niveau de la couche physique, des propriétés choisies de la requête de fichier initiale pour former les requêtes d'accès physique.

Pour l'exécution d'une requête d'accès physique donnée, le client de stockage 44 interagit avec le serveur de stockage 46 de la station de stockage qui contient l'espace de stockage 38 sur lequel est située l'adresse physique désignée par la requête d'accès physique concernée. Cette interaction sera précisée au moyen de la figure 5.

Comme on peut le voir sur la figure 5, l'exécution d'une requête d'accès physique par un client de stockage 44 comprend d'abord la réception de la requête d'accès physique par le serveur de stockage 46 considéré. Cette réception est ici réalisée sous la forme de l'envoi d'un en-tête ou "header" 62 qui indique au serveur de stockage 46 le type de requête, la taille de cette requête, et l'adresse physique qui sont visés.

La requête via son en-tête est alors stockée dans une file d'attente 64 du serveur de stockage 46. La file d'attente 64 comprend l'ensemble des requêtes d'accès non encore exécutées envoyées par l'ensemble des clients de stockage 44 au serveur de stockage 46 en question, ainsi que leur état d'exécution.

Un serveur de stockage 46 peut comporter plusieurs files d'attente 64, par exemple une file d'attente pour chaque client de stockage 44 du réseau, ou encore une file d'attente pour chaque espace de stockage dont le serveur de stockage 46 gère l'accès, ou tout autre agencement utile pour l'implémentation des stratégies d'ordonnancement qui seront décrites plus bas.

Le serveur de stockage 46 peut ainsi recevoir en cascade une quantité importante de requêtes d'un ou plusieurs clients de stockage et exécuter celles-ci dans l'ordre le plus favorable pour l'occupation de la station sur laquelle il s'exécute, l'occupation des disques qu'il gère, et l'occupation réseau en général.

Dans ce qui est connu, la relation client de stockage / serveur de stockage est dite "orientée client". Dans ce type de relation, c'est la file d'attente des requêtes du client de stockage 44 qui prévaut, et le client n'est autorisé à envoyer une nouvelle requête d'accès à un serveur que lorsque celui-ci a répondu à la requête précédente.

L'architecture décrite ici constitue une "orientation serveur" de la gestion de l'accès à l'espace de stockage. Contrairement à ce qui est connu, un client de stockage 44 donné peut ainsi envoyer une multitude de requêtes d'accès à un même serveur de stockage 46, sans que celui-ci n'ait à retourner d'abord le résultat d'une requête émise antérieurement par le client de stockage 44. Cela permet de mieux équilibrer la charge disque et réseau dans les accès entrée/sortie et est particulièrement avantageux.

Parallèlement à la réception des requêtes dans sa file d'attente 64, le serveur de stockage 46 effectue une étape 66 en boucle dans laquelle il ordonne et exécute les requêtes reçues dans la file d'attente. La requête correspondant à l'en-tête 62 est donc traitée dans cette boucle, dans l'ordre déterminé par le serveur de stockage.

Dans l'étape 66, le serveur exécute un ordonnancement des requêtes dans la file d'attente 64, afin d'optimiser localement l'utilisation des espaces de stockage dont il gère l'accès ainsi que l'utilisation de la station sur laquelle il s'exécute en prenant en compte des paramètres comme la charge processeur, l'occupation de la mémoire centrale de la station, etc.

L'ordonnancement et l'exécution effectués à l'étape 66 sont précisés à l'aide de la figure 6.

À un instant donné, le serveur de stockage 46 a dans sa file d'attente un ensemble de requêtes pouvant prendre divers états. Ces états peuvent être par exemple "à traiter" (lorsqu'une requête vient d'être reçue dans la file d'attente),

"en attente d'exécution" (lorsque le serveur de stockage 46 a l'ensemble des données nécessaires à l'exécution d'une requête et a programmé son exécution à un moment ultérieur), ou encore "en cours d'exécution".

Il apparaît par ailleurs que ces requêtes ont au moins deux natures distinctes. Une première nature est qualifiée de "réseau", et désigne un échange qui doit être réalisé entre le serveur de stockage 46 et un client de stockage 44 donné. L'autre nature est qualifiée de "disque" et désigne un accès que doit réaliser le serveur de stockage 46 sur l'un des espaces de stockage qu'il gère, pour lire ou écrire des données.

L'ordonnancement de l'étape 66 est réalisé sur la base de la nature de ces requêtes et de leur état, de paramètres d'état du réseau du système et de paramètres d'état des espaces de stockages gérés par le serveur de stockage 46, ainsi que l'utilisation de la station sur laquelle il s'exécute en prenant en compte des paramètres comme la charge processeur, l'occupation de la mémoire centrale de la station, etc.

La description de l'étape 66 va être faite dans le cas où le serveur de stockage gère plusieurs disques de stockage de données. Il en résulte que de nombreux éléments qui sont de nature globale par rapport à la boucle sont des tables, éventuellement multidimensionnelles.

Ainsi, il sera fait appel à un tableau Last_Sect, qui comporte une seule ligne, et dont chaque colonne désigne le dernier secteur accédé pour le disque correspondant à cette colonne.

De même, il sera fait appel à une matrice Tm_Used, dans laquelle les lignes désignent chacune un client de stockage, et les colonnes chacune un disque, les valeurs des éléments au croisement de la ligne x et de la colonne y représentant le temps d'occupation du disque y pour des requêtes émises par le client x.

La boucle de l'étape 66 traite des données 70. Les données 70 contiennent une liste de requêtes File et une liste de requêtes List_Req. La liste de requêtes File contient un ensemble de requêtes ayant un état "à traiter", c'est-à-dire les requêtes reçues instantanément dans la ou les files d'attente du serveur de stockage.

La liste List_Req contient un ensemble de requêtes ayant un état "en attente d'exécution" ou "en cours d'exécution". Ces requêtes sont chacune accompagnées d'un indicateur d'âge. L'indicateur d'âge d'une requête indique le nombre de boucles qui ont été parcourues depuis que cette requête a été ajoutée à la liste List_Req.

Dans une étape 72, le serveur de stockage appelle une fonction Init() avec pour argument les listes File et List_Req. La fonction Init() est décrite plus avant avec la figure 7.

La fonction Init() débute dans une étape 700, avec dans une étape 702 l'appel d'une fonction Add_New_Req() qui a pour arguments les listes File et List_Req. La fonction Add_New_Req() a pour rôle de prendre toutes les nouvelles requêtes de la liste File et les ajouter à la liste List_Req. Dans la liste List_Req, l'indicateur d'âge des nouvelles requêtes est initialisé à 0 par la fonction Add_New_Req().

L'étape 702 est suivie par une double condition portant sur l'occupation du serveur de stockage, afin d'optimiser le fonctionnement du système. La première condition est testée dans une étape 704, dans laquelle un indicateur d'attente Stat_Wt est testé.

Lorsque l'indicateur Stat_Wt est égal à 0, cela signifie qu'aucune attente n'a eu lieu lors de la boucle précédente. De manière inverse, une attente lors de la boucle précédente est indiquée par un indicateur Stat_Wt égal à 1.

La deuxième condition est testée dans une étape 706, dans laquelle le serveur de stockage vérifie qu'il y a plus de deux requêtes dans la liste File.

Si l'une de ces conditions n'est pas remplie, à savoir s'il y a eu attente lors de la boucle précédente, ou si plus de deux requêtes sont dans la liste File, alors la fonction Init() se poursuit dans une étape 708 dans laquelle l'indicateur Stat_Wt est mis à 0 pour la prochaine boucle.

Ensuite, dans une étape 710, le serveur de stockage teste si la liste List_Req est vide. Si ce n'est pas le cas, la fonction Init() se termine à l'étape 712, et la boucle d'ordonnancement peut se prolonger pour traiter les requêtes de la liste List_Req.

Si la liste List_Req est vide, alors il n'y a pas lieu de poursuivre la boucle d'ordonnancement, et le serveur de stockage attend une milliseconde par une fonction Wait(1) dans une étape 714, puis met l'indicateur Stat_Wt à 1 pour la prochaine boucle et reprend à l'étape 702, pour récupérer d'éventuelles nouvelles requêtes reçues par la ou les files d'attente du serveur de stockage.

Après la fonction Init(), le serveur de stockage appelle une fonction Run_Old() dans une étape 76. Cette fonction a pour but d'exécuter les requêtes de List_Req qui ont un indicateur d'âge très élevé.

La fonction Run_Old() est décrite au moyen de la figure 8, et retourne un indicateur Rst égal à 1 si une requête âgée est exécutée, et égal à 0 sinon.

Après une étape de départ 800, le serveur de stockage appelle dans une étape 802 une fonction Max_Age(). La fonction Max_Age() prend comme argument la liste List_Req et retourne l'indicateur d'âge le plus élevé des requêtes de List_Req.

Si cet indicateur d'âge est supérieur à 150, alors dans une étape 804, le serveur de stockage appelle une fonction Age() qui prend comme arguments la liste List_Req et le nombre 120. La fonction Age() détermine l'ensemble des requêtes de List_Req qui ont un indicateur d'âge supérieur à 120. Ces requêtes sont stockées dans une liste de requêtes List_Old.

Ensuite, dans une étape 806, le serveur de stockage appelle une fonction Req_Min_Sect() avec la liste List_Old et le tableau Last_Sect comme arguments. La fonction Req_Min_Sect() permet de déterminer quelle est la requête parmi la liste List_Old qui présente le secteur d'accès de requête le plus proche du dernier secteur accédé récemment.

Cela est réalisé en calculant, pour chaque requête contenue dans List_Old, la valeur absolue de la distance entre le secteur de disque visé et le dernier secteur accédé de ce disque, comme contenu dans Last_Sect. Une fois que le minimum est déterminé, la requête correspondante est stockée dans une requête Req.

Ensuite, le serveur de stockage exécute la requête Req en l'appelant comme argument d'une fonction Exec() dans une étape 808. La fonction Exec() exécute la requête Req, mesure le temps d'exécution de cette requête et stocke ce temps dans un nombre T_ex.

L'exécution d'une requête est décrite à l'aide de la figure 9. Cette exécution est basée sur un triplet adresse physique – taille – type de requête 900 que contient l'en-tête dans la file d'attente 64.

Dans une étape 902, un test sur le type de la requête détermine la chaîne des entrées/sorties disques et réseau à effectuer.

Si c'est une requête en écriture, le serveur de stockage demande au client de stockage de lui envoyer les données d'écriture dans une étape 904. Le serveur

de stockage attend les données, et, à réception, les inscrit dans l'espace désigné par l'adresse physique dans une étape 906.

Le serveur de stockage 46 émet alors un accusé de réception d'écriture 908 au client de stockage 44 afin de confirmer l'écriture. Après cela, l'exécution se termine dans une étape 914.

Si c'est une requête en lecture, le serveur de stockage 46 accède aux données contenues dans l'espace désigné par l'adresse physique dans une étape 910, jusqu'à la taille de la requête, et transmet celles-ci au client de stockage 44 dans une étape 912. Après cela, l'exécution se termine dans une étape 914.

Une fois la requête Req exécutée, le serveur de stockage met à jour la liste List_Req dans une étape 810. Cette mise à jour est réalisée en appelant une fonction Upd() avec comme arguments la liste List_Req et le nombre T_ex.

Cette fonction retire la requête Req des listes List_Req et List_Old, et met à jour une matrice Tm_Used, en ajoutant le nombre T_ex à l'élément au croisement de la ligne qui correspond au client de stockage 44 qui a émis la requête Req, et de la colonne qui correspond au disque visé par la requête Req. Cela permet de tenir à jour l'occupation de chaque disque par chaque client de stockage. Enfin, le tableau Last_Sect est mis à jour à la colonne du disque qui a été accédé par la requête Req, pour tenir compte du dernier secteur réellement accédé.

Le serveur de stockage 46 teste ensuite dans une étape 812 si la liste List_Old est vide. Si c'est le cas, alors l'indicateur Rst est mis à 1 dans une étape 814 pour indiquer qu'une requête "âgée" a été exécutée et la fonction Run_Old() se termine dans une étape 816. Si ce n'est pas le cas, la fonction Run_Old() retourne à l'étape 806 pour exécuter les autres requêtes âgées restantes.

Dans le cas où la fonction Max_Age() retourne un indicateur d'âge inférieur à 150, alors l'indicateur Rst est mis à 0 dans une étape 818 pour indiquer

qu'aucune requête "âgée" n'a été exécutée, et la fonction Run_Old() se termine dans une étape 820.

La boucle d'ordonnancement et d'exécution se poursuit ensuite par un test sur l'indicateur Rst dans une étape 78, afin de déterminer si une requête "âgée" a été exécutée. Si c'est le cas, alors le serveur de stockage réitère la boucle avec l'étape 72 en réappelant la fonction Init().

Sinon, le serveur de stockage termine la boucle d'ordonnancement et d'exécution en appelant une fonction Run_Min_Use() avec comme argument la liste List_Req.

La fonction Run_Min_Use() est décrite à l'aide de la figure 10. Après initialisation dans une étape 1000, le serveur de stockage 46 appelle une fonction Add_Age() avec comme argument le nombre 1 dans une étape 1002. La fonction Add_Age() incrémente de 1 l'indicateur d'âge de toutes les requêtes de la liste List_Req, et initialise un compteur Min_t à 0.

Dans une étape 1004, le serveur de stockage 46 appelle ensuite une fonction Use_Inf_Min_t() avec comme arguments la liste List_Req et le compteur Min_t. La fonction Use_Inf_Min_t() parcourt la liste List_Req, et vérifie pour chaque requête si l'élément de la matrice Tm_Used au croisement de la ligne correspondant au client de stockage 44 qui l'a émise et de la colonne correspondant au disque qu'elle désigne est inférieur à Min_t.

Concrètement, cela signifie qu'une requête donnée est sélectionnée si le client qui l'a émise a déjà occupé le disque qu'elle vise pendant un temps inférieur à Min_t. Toutes les requêtes ainsi sélectionnées sont stockées dans une liste List_Min_Req.

Dans une étape 1006, le serveur de stockage teste si List_Min_Req est vide. Si c'est le cas, alors le compteur Min_t est incrémenté de 1 dans une étape 1008, et l'étape 1004 est répétée.

Une fois que la liste List_Min_Req contient au moins une requête, le serveur de stockage 46 exécute des étapes 1010, 1012 et 1014 qui ne diffèrent des étapes 906, 908, et 910 précédemment décrites qu'en ce que c'est la liste List_Min_Req qui est ici utilisée, au lieu de la liste List_Old.

Après l'exécution de la requête la plus favorable selon les étapes 1010, 1012 et 1014, le serveur de stockage 46 appelle une fonction Rst_Tm_Used() dans une étape 1016.

La fonction Rst_Tm_Used() a pour but de réinitialiser la matrice Tm_Used dans le cas où un client de stockage 44 a beaucoup utilisé les disques par rapport aux autres clients de stockage.

Pour cela, la fonction Rst_Tm_Used() additionne tous les éléments de la matrice Tm_Used. Cela représente la somme totale des temps d'occupation des disques gérés par le serveur de stockage 46, par la totalité des clients de stockage 44.

Si cette somme totale excède une valeur prédéterminée, alors tous les éléments de la matrice Tm_Used sont mis à 0. Sinon, la matrice Tm_Used est inchangée.

Après l'étape 1016, la fonction Run_Min_Use() se termine dans une étape 1018, et la boucle d'ordonnancement et d'exécution est recommencée à l'étape 72.

La fonction Run_Min_Use() permet donc d'ordonner l'exécution des requêtes sur la base d'informations contenues dans l'en-tête des requêtes,

indépendamment de la présence des données éventuellement désignées par ces requêtes.

Il est donc ainsi possible d'ordonner l'exécution d'une quantité importante de requêtes, notamment des requêtes en écriture, sans surcharger l'espace mémoire avec les données à écrire de ces requêtes.

Dans d'autres applications, il serait néanmoins envisageable de n'ordonner que les requêtes de la liste File pour lesquelles l'ensemble des données nécessaires à l'exécution de la requête sont disponibles. Cela pourrait être fait en assurant en parallèle une boucle d'alimentation en données, assurant ainsi que l'espace alloué au stockage des données de requête est rempli de manière à optimiser la quantité de requêtes à ordonner.

La description de l'étape 66 a été faite dans le cas où le serveur de stockage gère plusieurs disques de stockage de données.

Il en résulte que de nombreux éléments qui sont de nature globale par rapport à la boucle ont été des tables, parfois multidimensionnelles. Dans le cas où le serveur de stockage ne gère qu'un seul disque, la situation peut être simplifiée. Alors, l'élément Last_Sect devient une valeur simple, et l'élément Tm_Used un tableau à une dimension (les clients de stockage).

Par ailleurs, l'ordonnancement a été réalisé ici en plaçant toutes les requêtes des files d'attente du serveur de stockage ensemble. Cependant, il serait possible de distinguer les requêtes en fonction de la file d'attente dont elles sont issues respectivement, soit en indiciant la liste List_Req, soit en exécutant un ordonnancement pour chaque file d'attente, en série ou en parallèle.

On a représenté sur la figure 11 une boucle d'ordonnancement et d'exécution en variante.

Dans ce mode de réalisation particulier, la boucle d'ordonnancement et d'exécution est essentiellement identique à celle présentée sur la figure 6, à ceci près qu'elle comporte en plus une boucle d'endormissement 110 qui est exécutée préalablement aux étapes représentées sur la figure 6.

La boucle d'endormissement 110 comporte une fonction de gestion d'endormissement Sleep_Mng() 112 dont le résultat est stocké dans une variable Slp.

La variable Slp indique la décision d'un endormissement temporaire du serveur de stockage ou non. Cette fonction sera décrite plus avant avec les figures 13 à 15.

Après la fonction Sleep_Mng() 112, la boucle d'endormissement 110 comporte un test 114 portant sur la valeur de la variable Slp.

Dans le cas où cette variable est non nulle, alors un endormissement temporaire du serveur de stockage est réalisé par une fonction Force_Slp() 116. Ensuite la boucle d'endormissement 110 est réinitialisée en 112.

La fonction Force_Slp() 116 "endort" le serveur de stockage en émettant dans la file d'attente une requête dite d'endormissement. La requête d'endormissement a priorité sur toutes les autres requêtes. Lorsqu'elle est exécutée, elle fait tourner le serveur de stockage à vide pendant une durée paramétrable. On peut voir cette fonction comme l'équivalent de la fonction Wait() de la figure 7.

Dans le cas où la variable Slp est nulle, la boucle d'ordonnancement et d'exécution s'exécute exactement comme celle représentée sur la figure 6.

La fonction Slp_Mng() va maintenant être décrite à l'aide de la figure 12. Comme on peut le voir sur cette figure, la fonction Slp_Mng() comporte l'exécution séquentielle d'une fonction Upd_Slp_Par() 1122, d'une fonction

Perf_Slp() 1124, et d'une fonction Mnt_Slp() 1126, avant de se terminer en 1128.

La fonction Upd_Slp_Par() 1122 a pour rôle de mettre à jour les paramètres utilisés pour décider de l'endormissement ou non. Cette fonction va maintenant être décrite à l'aide de la figure 13.

Comme on peut le voir sur la figure 13, la fonction Upd_Slp_Par() 1122 met à jour deux paramètres Tm_psd, Nb_Rq_Slp et la variable Slp.

Dans une étape 1302, le paramètre Tm_psd est mis à jour avec une fonction Elps_Time(). La fonction Elps_Time() calcule combien de temps s'est écoulé depuis la dernière exécution de la fonction Upd_Slp_Par() 1122.

Cela peut être réalisé, par exemple, en gardant de boucle en boucle une variable horodateur mise à jour à chaque exécution, et en comparant cette variable au temps courant à l'exécution lors de la boucle suivante.

Dans une étape 1304, le paramètre Nb_Rq_Slp est incrémenté par une valeur Tm_psd*Fq_Rq_Slp. Le paramètre Nb_Rq_Slp représente un nombre de requêtes d'endormissement.

Dans le mode de réalisation décrit ici, il existe deux principaux types de conditions d'endormissement. Le premier est un type de conditions reliées à la performance. Le deuxième est un type relatif à un taux nominal d'occupation. Ce taux peut notamment être défini par le biais du module d'administration ou d'une manière générale être vu comme un paramètre fixé par l'administrateur du système.

Le paramètre Nb_Rq_Slp relève de ce deuxième type. C'est un compteur qui prévoit que le serveur crée des requêtes d'endormissement avec une fréquence

Fq_Rq_Slp qui est un paramètre réglé par l'administrateur du serveur de stockage.

Cependant, comme cela apparaîtra plus bas, les requêtes d'endormissement ne sont réellement exécutées que sous certaines conditions. Ce compteur permet de déterminer combien de requêtes d'endormissement auraient pu être exécutées.

Ensuite, dans une étape 1306, la variable Slp est réinitialisée à 0 pour la boucle d'endormissement courant, et la fonction Upd_Par_Slp() se termine en 1308.

La fonction Perf_Slp() va maintenant être décrite à l'aide de la figure 14. Cette fonction permet de décider un endormissement sur la base des paramètres d'état du nœud stockant et de la file d'attente.

Pour cela, cette fonction repose sur deux tests 1402 et 1404. Le premier test 1402 porte sur l'occupation des ressources locales, c'est-à-dire les ressources du nœud stockant sur lequel s'exécute le serveur de stockage.

Dans le mode de réalisation décrit ici, ce sont les ressources processeurs qui sont testées. Pour cela, des fonctions d'évaluations du taux d'occupation du processeur sont appelées. Ces fonctions peuvent être du type standard et reposer par exemple sur la consultation de variables globales maintenues par le système d'exploitation, ou bien être des fonctions plus spécifiques.

Si le processeur est déjà extrêmement chargé (au dessus de 90% par exemple), alors il est décidé d'endormir le serveur de stockage pour que celui-ci ne dégrade pas les performances du nœud stockant.

Ainsi, si c'est le cas, la variable Slp est mise à 1 en 1406 et la fonction Perf_Slp() se termine en 1408.

On notera que de nombreuses autres conditions peuvent être utilisées ici en combinaison avec la charge processeur, ou à la place de celle-ci, comme la charge d'accès de l'unité de mémoire locale par exemple ou d'autres que l'homme du métier saura envisager.

Le deuxième test 1404 n'est réalisé que si le premier test 1402 est négatif, c'est-à-dire si la charge processeur n'est pas trop importante. Ce deuxième test porte sur l'évaluation du nombre de requêtes contenues dans la file d'attente.

En effet, si ce nombre est trop faible, le serveur de stockage ne tire pas parti de la boucle d'ordonnancement, ce qui réduit potentiellement les performances.

Dès lors, si le nombre de requêtes présentes dans la file d'attente est trop faible, la variable Slp est mise à 1 en 1406, et la fonction Perf_Slp() se termine en 1408. Sinon, la fonction se termine directement en 1408.

On notera qu'ici aussi de nombreuses autres conditions peuvent être utilisées, le principe étant d'endormir le serveur de stockage tant que des conditions de performance favorables ne sont pas remplies.

Parmi ces conditions alternatives, on notera par exemple le type des requêtes présentes dans la file d'attente. Ainsi, si les requêtes sont "lointaines", c'est-à-dire si la distance entre l'adresse physique désignée par chaque requête et une adresse physique accédée précédemment par le serveur de stockage est supérieure à un seuil fixé, on peut considérer comme plus favorable d'attendre qu'une requête plus "proche" arrive dans la file d'attente. On pourrait également utiliser comme critère de type la nature des requêtes, c'est-à-dire lecture ou écriture ou d'autres critères relatifs aux caractéristiques des requêtes.

La fonction Mnt_Slp() va maintenant être décrite à l'aide de la figure 15. Cette fonction permet d'annuler un endormissement qui était prévu ou au contraire

d'imposer un endormissement de "maintenance". La fonction Mnt_Slp() est basée sur deux tests 1502 et 1508.

Le premier test 1502 compare le paramètre Nb_Rq_Slp à un nombre minimal de requêtes d'endormissement à avoir pour permettre l'exécution d'une d'entre elles. Cela revient à déterminer si beaucoup de requêtes d'endormissement ont été exécutées récemment.

Si le paramètre Nb_Rq_Slp est inférieur au nombre minimal, alors la variable Slp est mise à 0 en 1504 et la fonction se termine en 1506.

Le deuxième test 1508 n'est réalisé que si le premier test 1502 est positif. Ce deuxième test compare le paramètre Nb_Rq_Slp à un nombre maximal de requêtes d'endormissement. Cela revient à déterminer si cela fait très longtemps qu'aucune requête d'endormissement n'a été exécuté.

Si le paramètre Nb_Rq_Slp est inférieur au nombre maximal, la fonction se termine en 1506. Dans le cas contraire, la variable Slp est mise à 1 en 1510, puis la fonction se termine en 1506.

Cela signifie que :

- même si a priori une exécution d'une requête d'endormissement est prévue, on ne va pas le faire car beaucoup d'autres requêtes ont déjà été exécutées il y a peu de temps, ou qu'au contraire,

- lorsqu'un certain temps s'est écoulé et qu'aucune requête d'endormissement n'a été exécutée, on décide arbitrairement d'en exécuter une, même si cela n'est pas nécessaire au vu des autres critères.

Du fait que la fonction Mnt_Slp() s'exécute après la fonction Perf_Slp(), on comprend bien qu'elle peut retourner les décisions de cette dernière. C'est-à-

dire que, pour des raisons de maintenance du serveur de stockage, cette fonction permet d'annuler un endormissement prévu ou d'en forcer un non-prévu, en jouant sur la variable Slp.

On notera par ailleurs que la fonction Force_Slp() décrémente d'une unité le compteur Nb_Rq_Slp à chacune de ses exécutions.

Comme il apparaît à la lecture de ce qui précède, la boucle d'ordonnancement et d'exécution comprend trois parties principales traitées en série et une pré-boucle optionnelle :

- une boucle d'endormissement optionnelle pour garantir les performances du nœud stockant sur lequel réside le serveur de stockage ;

- une première partie de gestion des nouvelles requêtes ;

- une deuxième partie de traitement des requêtes les plus âgées ;

- une troisième partie de traitement des requêtes issues des clients de stockage qui ont le moins utilisé le serveur de stockage.

Il apparaît clairement que ces parties sont indépendantes les unes des autres et qu'une boucle simplifiée pourrait ne contenir qu'une seule ou plusieurs de celles-ci. Il apparaît également clair que le traitement de ces parties pourrait être parallèle, et qu'au lieu de réinitialiser la boucle après l'exécution des requêtes "âgées", il serait possible de réaliser la troisième partie.

Les deuxièmes et troisièmes parties doivent être perçues comme des exemples particuliers de concepts plus généraux.

Ainsi, dans la deuxième partie, le traitement des requêtes "âgées" relève d'un souci de gérer les requêtes "exceptionnelles", qui, pour une quelconque raison,

ne sont pas exécutées en priorité par l'algorithme. C'est cette idée directrice qu'il convient de retenir, en ce sens que d'autres implémentations pour éviter de tels cas pourraient être envisagés.

En ce qui concerne la troisième partie, le concept général est d'ordonner les requêtes en fonction d'un critère quantitatif basé sur la relation entre le client de stockage et le serveur de stockage. Ainsi, dans l'exemple décrit, un critère quantitatif de temps d'utilisation des unités de mémoire locales est utilisé pour discriminer les requêtes entre elles.

Il serait cependant possible d'utiliser d'autres critères quantitatifs, basés sur des statistiques caractérisant les interactions client de stockage – serveur de stockage, comme le taux moyen d'échange de données, la latence réseau moyenne constatée lors de ces interactions, le taux de perte de paquets, etc.

Par ailleurs, l'implémentation ici décrite est donnée à titre d'exemple simplifié, et elle pourrait être améliorée encore par l'utilisation de techniques de programmation classiques, comme l'usage de tampons, ou encore la prise en compte d'autres paramètres pour l'ordonnancement.

Dans ce qui a été présenté, l'ordonnancement est basé sur une stratégie favorisant deux axes principaux : l'exécution des requêtes âgées, et le partage de la charge entre les disques et les clients.

D'autres stratégies peuvent être implémentées (en changeant en correspondance ce qui précède) pour favoriser d'autres approches comme :

- la maximisation de l'exploitation de la bande passante disque, par exemple en agrégeant les requêtes qui sont contiguës ou presque en une seule requête, ce qui permet d'économiser un accès disque ;

- la maximisation de l'exploitation de la latence disque, par exemple en générant au niveau du serveur de stockage des requêtes d'optimisation visant le centre du ou des disques pour diminuer la latence, ou en générant des requêtes prédictives (c'est-à-dire visant des données en prévision d'une future requête) au niveau du serveur de stockage.

D'autres stratégies et leur implémentation, ainsi que de nombreuses variantes apparaîtront de manière évidente à l'homme du métier.

Ainsi, l'application qui accède aux données stockées peut comporter un pilote qui gère les relations entre les divers éléments telle l'interaction application – système de fichiers, l'interaction système de fichiers – module de correspondance, l'interaction module de correspondance – client de stockage, implémentation de la stratégie du serveur de stockage en obtenant de chaque élément un résultat et en appelant l'élément suivant avec ce résultat (ou une forme modifiée de ce résultat).

En variante, le système est autonome et ne dépend pas de l'application qui appelle les données, et les éléments sont capables de communiquer entre eux, de sorte que l'information descend puis remonte les couches d'élément en élément.

De même, les communications entre ces éléments peuvent être assurées de différentes manières, par exemple au moyen de l'interface POSIX, des protocoles IP, TCP, UDP, d'une mémoire partagée, d'une RDMA (Remote Direct Access Memory). Il convient de garder à l'esprit que le but de l'invention est d'offrir les avantages de systèmes de stockage spécialisés sur la base des ressources réseaux existantes.

Un exemple de réalisation du système décrit ci-dessus est basé sur un réseau dans lequel les stations sont réalisées avec des ordinateurs comprenant :

* un processeur spécialisé ou généraliste (par exemple du type CISC ou RISC ou autre),

* un ou plusieurs disques de stockage (par exemple des disques durs à interface Serial ATA, ou SCSI, ou autre) ou tout autre type de stockage, et

* une interface réseau (par exemple Gigabit, Ethernet, Infiniband, SCI...)

* un environnement d'application basé sur un système d'exploitation (par exemple Linux) pour supporter des applications et offrir un gestionnaire de système de fichiers,

* un ensemble applicatif pour réaliser le module de correspondance, par exemple le module Clustered Logical Volume Manager de l'application Exanodes (marque déposée) de la société Seanodes (marque déposée),

* un ensemble applicatif pour réaliser le client de stockage et le serveur de stockage de chaque NBD, par exemple le module Exanodes Network Block Device de l'application Exanodes (marque déposée) de la société Seanodes (marque déposée),

* un ensemble applicatif pour gérer les éléments répartis, par exemple le module Exanodes Clustered Service Manager de l'application Exanodes (marque déposée) de la société Seanodes (marque déposée).

Ce type de système peut être réalisé dans un réseau comportant :

* des stations utilisateurs classiques, adaptées à un usage applicatif sur un réseau et qui jouent le rôle de nœuds applicatifs, et

* un ensemble de dispositifs informatiques réalisés conformément à ce qui précède, et qui jouent le rôle de serveurs du réseau et de nœuds de stockage.

D'autres matériels et applications apparaîtront à l'homme du métier pour réaliser des dispositifs en variante dans le cadre de l'invention.

L'invention englobe le système informatique comprenant les nœuds applicatifs et les nœuds stockant dans leur ensemble. Elle englobe également les éléments individuels de ce système informatique, et notamment les nœuds applicatifs et les nœuds stockants dans leur individualité, ainsi que les divers moyens pour les réaliser.

De même, le procédé de gestion de données est à considérer dans sa globalité, c'est-à-dire dans l'interaction des nœuds applicatifs et des nœuds stockants, mais également dans l'individualité des postes informatiques adaptés pour réaliser les nœuds applicatifs et les nœuds stockants de ce procédé.

La description qui précède a pour but de décrire un mode de réalisation particulier de l'invention. Elle ne saurait être considérée comme limitant ou décrivant celle-ci de manière limitative, et couvre notamment l'ensemble des combinaisons entre elles des caractéristiques des variantes décrites.

L'invention couvre également, en tant que produits, les éléments logiciels décrits, mis à disposition sous tout “medium” (support) lisible par ordinateur. L'expression “medium lisible par ordinateur” comprend les supports de stockage de données, magnétiques, optiques et/ou électroniques, aussi bien qu'un support ou véhicule de transmission, comme un signal analogique ou numérique.

De tels media couvrent aussi bien les éléments logiciels en eux même, c'est-à-dire les éléments propres à être exécutés directement, que les éléments logiciels qui servent à l'installation et/ou le déploiement, comme lorsque un disque d'installation ou un programme d'installation téléchargeable. Une telle installation peut être réalisée de manière globale, sur des postes clients et des postes serveurs, ou de manière séparée, avec à chaque fois des produits appropriés.

Revendications

1. Système informatique comprenant plusieurs postes informatiques dits noeuds interconnectés en un réseau,

- certains au moins des nœuds, dits stockants, comprenant au moins une unité de mémoire locale (38) à accès direct, et un serveur de stockage (46) agencé pour gérer l'accès à cette unité de mémoire locale (38) sur la base d'une file d'attente (64) de requêtes d'accès,

- certains au moins des nœuds, dits applicatifs, comprenant chacun :

* un environnement d'application comportant une représentation de fichiers accessibles depuis ce nœud sous forme d'adresses de blocs désignant chacune au moins une adresse physique sur une unité de mémoire locale (38) d'un noeud stockant, et

* un client de stockage (44) capable d'interagir avec un quelconque des serveurs de stockage (46), sur la base d'une requête d'accès désignant une adresse de blocs,

caractérisé en ce que l'un au moins des serveurs de stockage (46) comporte un ordonnanceur capable d'exécuter les requêtes d'accès contenues dans sa file d'attente (64) dans un ordre déterminé, et en ce que l'ordonnanceur est agencé pour déterminer cet ordre en fonction d'un jeu de règles formant critères de performance, et impliquant un ou plusieurs paramètres d'état de la file d'attente et/ou du nœud stockant sur lequel il réside.

2. Système informatique selon la revendication 1, caractérisé en ce que le jeu de règles implique en outre un ou plusieurs paramètres d'état de l'unité de mémoire locale.

3. Système informatique selon la revendication 1 ou 2, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) et basée sur le contenu d'un en-tête (62) de chaque requête, et en ce que cette règle est opératoire en l'absence d'autres données associées à ces requêtes.

4. Système informatique selon l'une des revendications précédentes, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) et basée sur un critère quantitatif établi sur la base d'interactions antérieures de ce client de stockage (44) avec le serveur de stockage (46) concerné.

5. Système informatique selon la revendication 4, caractérisé en ce que le critère quantitatif est établi sur la base de la durée de sollicitation antérieure du serveur de stockage (46) concerné par le client de stockage.

6. Système informatique selon l'une des revendications précédentes, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) et basée sur la durée de présence des requêtes dans la file d'attente (64).

7. Système informatique selon l'une des revendications précédentes, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) et basée sur la distance entre l'adresse physique désignée par chaque requête et une adresse physique accédée précédemment par le serveur de stockage (46).

8. Système informatique selon l'une des revendications précédentes, caractérisé en ce que l'un au moins des serveurs de stockage (46) est agencé pour exécuter les requêtes d'accès contenues dans sa file d'attente (64) dans un ordre déterminé en fonction d'un jeu de règles choisi parmi une pluralité de jeux de règles différents formant stratégies quant aux critères de performance.

9. Système informatique selon l'une des revendications précédentes, caractérisé en ce que certains au moins des clients de stockage (44) sont autorisés à émettre sélectivement une requête d'accès à un serveur de stockage donné, avant la complétion d'une requête d'accès à ce serveur de stockage (46), émise antérieurement par le client de stockage considéré.

10. Système informatique selon l'une des revendications précédentes, caractérisé en ce que certains au moins des serveurs de stockage (46) gèrent plusieurs unités de mémoire locale (38).

11. Système informatique selon l'une des revendications précédentes, caractérisé en ce que certains au moins des nœuds sont des nœuds applicatifs et des nœuds stockants.

12. Système informatique selon l'une des revendications précédentes, caractérisé en ce que le jeu de règles comprend une règle propre à décaler l'exécution des requêtes contenues dans la file d'attente d'une durée choisie.

13. Système informatique selon la revendication 12, caractérisé en ce que ladite règle est basée sur un taux d'occupation des ressources du nœud stockant.

14. Système informatique selon la revendication 12 ou 13, caractérisé en ce que ladite règle est basée sur un critère choisi parmi le groupe comprenant le nombre de requêtes contenues dans la file d'attente, et le type des requêtes contenues dans la file d'attente.

15. Système informatique selon l'une des revendications 12 à 14, caractérisé en ce que ladite règle est basée sur l'évaluation d'un temps écoulé depuis le précédent décalage.

16. Système informatique selon l'une des revendications précédentes, caractérisé en ce qu'il comprend un module de correspondance (36), capable de maintenir une correspondance entre chaque adresse de blocs et au moins une adresse physique sur une unité de mémoire locale (38) d'un noeud stockant.

17. Système informatique selon la revendication 16, caractérisé en ce que le module de correspondance (36) comprend une table de correspondance (42) comprenant une correspondance entre chaque adresse de blocs et au moins une adresse physique sur une unité de mémoire locale (38) d'un noeud stockant, et un moteur (40) agencé pour définir au moins une adresse physique pour une requête donnée en interrogeant la table de correspondance (42) avec une adresse de blocs, et de transmettre la requête donnée avec la ou les adresses physiques déterminées au client de stockage (44) concerné pour exécution de la requête donnée.

18. Procédé de gestion de données, applicable dans un réseau comprenant plusieurs postes informatiques interconnectés, dits noeuds, comportant les étapes suivantes :

a. émettre une requête de fichier depuis un nœud applicatif du réseau, sur la base d'une représentation sous forme d'adresses virtuelles,

b. sur la base d'une correspondance entre chaque adresse virtuelle et au moins une adresse physique sur une unité de mémoire locale (38) d'un noeud stockant du réseau, déterminer au moins une adresse physique correspondant à ladite adresse virtuelle,

c. émettre une requête d'accès désignant ladite adresse physique depuis un client de stockage (44) vers un serveur de stockage (46) gérant l'accès à l'unité de mémoire locale (38) associée à ladite adresse physique,

caractérisé en ce que le procédé comporte en outre l'étape suivante :

d. placer la requête d'accès dans une file d'attente (64) dudit serveur de stockage (46), et exécuter les requêtes d'accès contenues dans ladite file d'attente (64), dans un ordre déterminé en fonction d'un jeu de règles formant critères de performance, et impliquant un ou plusieurs paramètres d'état de la file d'attente et/ou du nœud stockant sur lequel il réside.

19. Procédé selon la revendication 18, caractérisé en ce que le jeu de règles implique en outre un ou plusieurs paramètres d'état de l'unité de mémoire locale.

20. Procédé selon la revendication 18 ou 19, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) et basée sur le contenu d'un en-tête (62) de chaque requête, et en ce que cette règle est opératoire en l'absence d'autres données associées à ces requêtes.

21. Procédé selon l'une des revendications 18 à 20, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) et basée sur un critère quantitatif établi sur la base d'interactions antérieures du client de stockage (44) émettant chaque requête avec le serveur de stockage (46) concerné.

22. Procédé selon la revendication 21, caractérisé en ce que le critère quantitatif est établi sur la base de la durée de sollicitation antérieure du serveur de stockage (46) concerné par le client de stockage.

23. Procédé selon l'une des revendications 18 à 22, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes dans la file d'attente (64) depuis un temps choisi.

24. Procédé selon l'une des revendications 18 à 23, caractérisé en ce que le jeu de règles comprend une règle propre à la sélection de requêtes présentes

dans la file d'attente (64) et basée sur la distance entre l'adresse physique désignée par chaque requête et une adresse physique accédée précédemment par le serveur de stockage (46).

25. Procédé selon l'une des revendications 18 à 24, caractérisé en ce que le jeu de règles comprend plusieurs règles choisies parmi une pluralité de jeux de règles différents formant stratégies quant aux critères de performance.

26. Procédé selon l'une des revendications 16 à 23, caractérisé en ce que l'étape c. peut être reproduite par un même émetteur de requête d'accès (44) avec une autre adresse physique, avant la complétion de l'étape d. pour une requête d'accès à ce serveur de stockage (46), émise antérieurement par l'émetteur de requête d'accès (44) considéré.

27. Procédé selon l'une des revendications 18 à 26, caractérisé en ce que certains au moins des serveurs de stockage (46) comportent une file d'attente (64) pour chaque émetteur (44) de requête d'accès.

28. Procédé selon l'une des revendications 18 à 27, caractérisé en ce que caractérisé en ce que certains au moins des serveurs de stockage (46) gèrent l'accès à plusieurs unités de mémoire locale (38).

29. Procédé selon l'une des revendications 18 à 28, caractérisé en ce que le jeu de règles comprend une règle propre à décaler l'exécution des requêtes contenues dans la file d'attente d'une durée choisie.

30. Procédé selon la revendication 29, caractérisé en ce que ladite règle est basée sur un taux d'occupation des ressources du nœud stockant.

31. Procédé selon la revendication 29 ou 30, caractérisé en ce que ladite règle est basée sur un critère choisi parmi le groupe comprenant le nombre de

requêtes contenues dans la file d'attente, et le type des requêtes contenues dans la file d'attente.

32. Procédé selon l'une des revendications 29 à 31, caractérisé en ce que ladite règle est basée sur l'évaluation d'un temps écoulé depuis le précédent décalage.

33. Dispositif informatique formant nœud stockant agencé pour effectuer l'étape d. du procédé selon l'une des revendications 18 à 32.

34. Produit de programme d'ordinateur comportant des moyens de code de programme propres à mettre en œuvre le procédé selon l'une des revendications 18 à 33 lorsqu'il est exécuté sur un ordinateur.

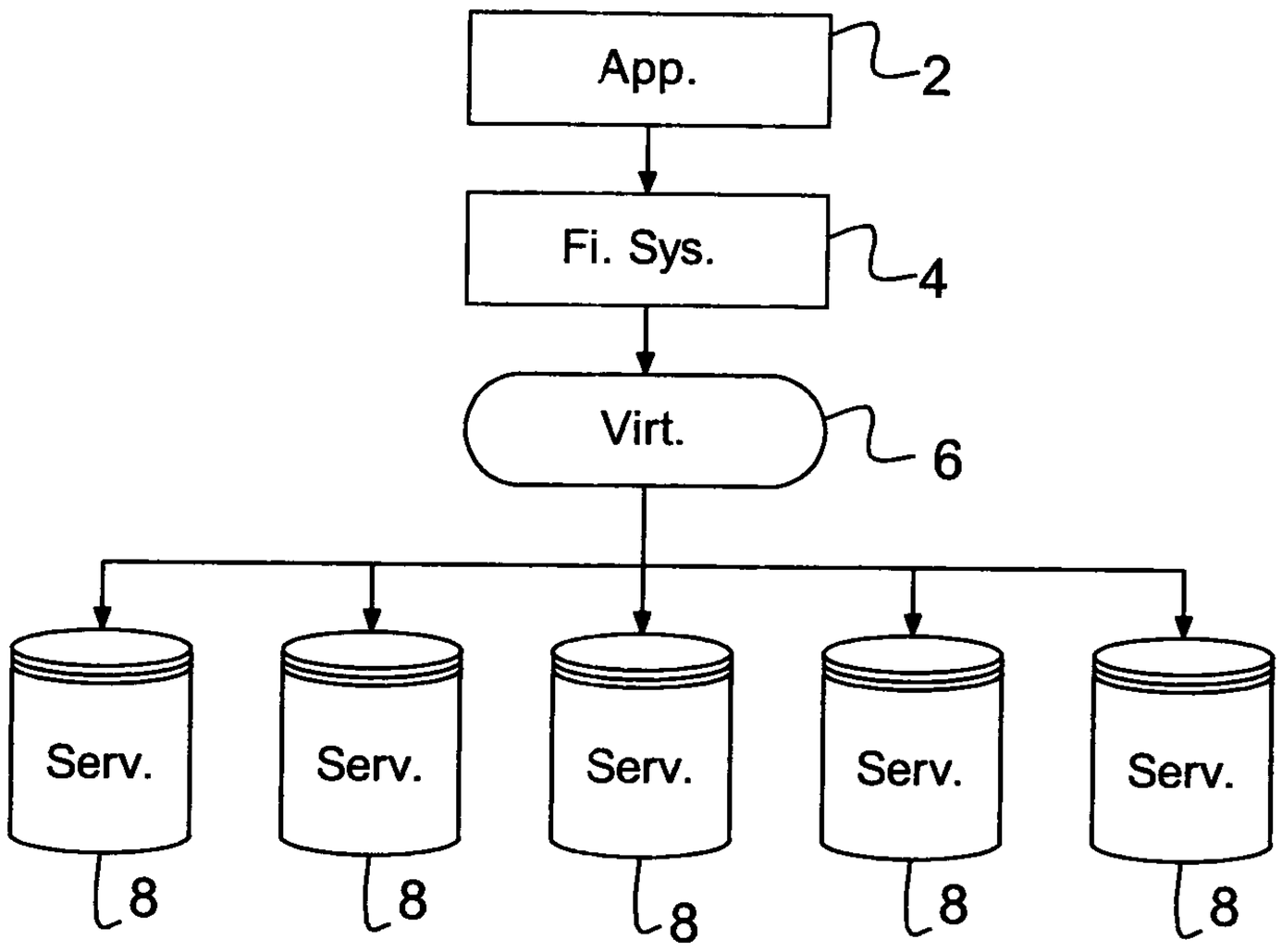

App.
2
Fi. Sys.
4
Virt.
6
Serv.
Serv.
Serv.
Serv.
Serv.
8
8
8
8
8

Fig. 1

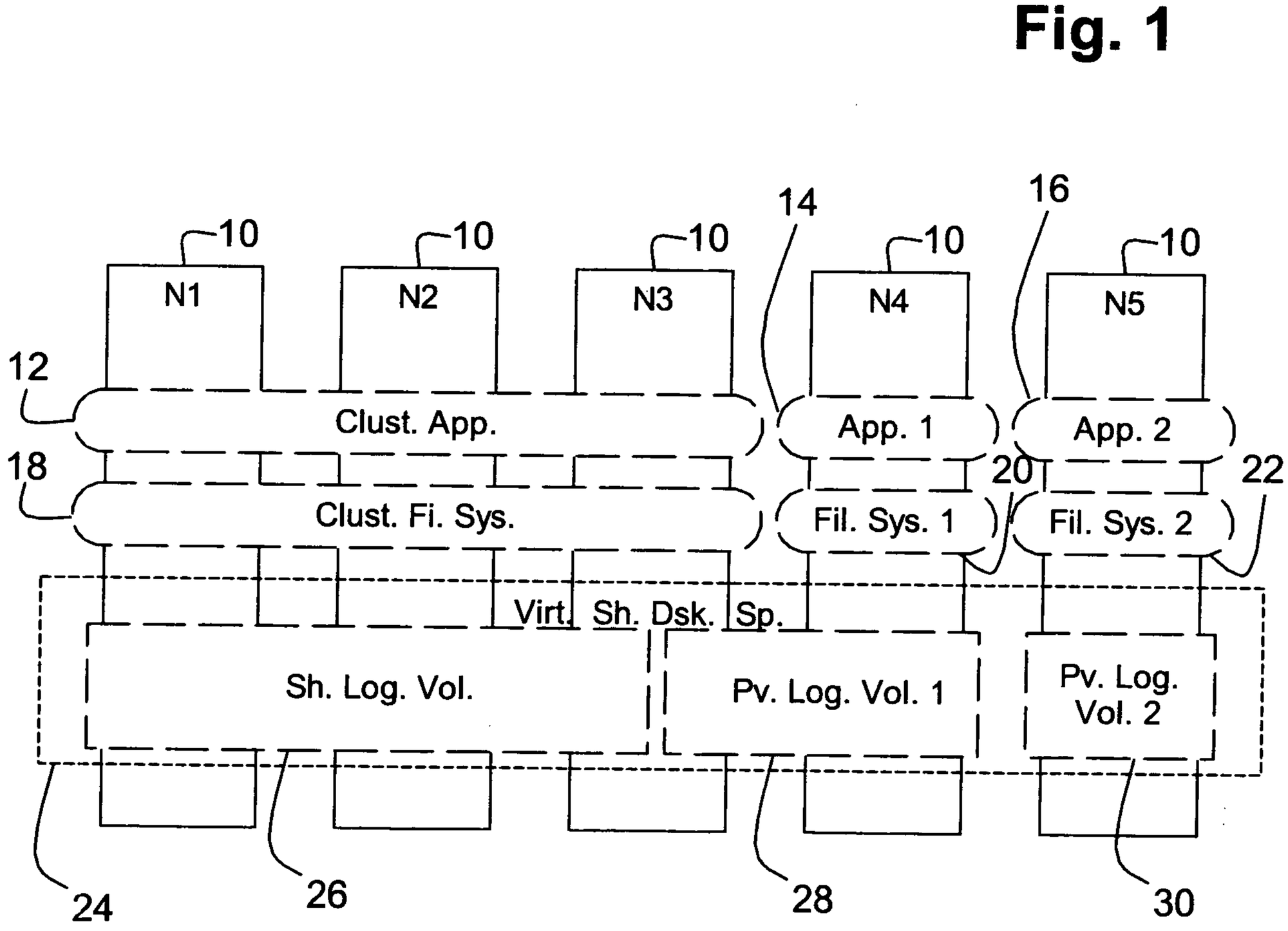

10
10
10
10
10
N1
N2
N3
N4
N5
12
14
16
Clust. App.
App. 1
App. 2
18
20
22
Clust. Fi. Sys.
Fil. Sys. 1
Fil. Sys. 2
Virt. Sh. Dsk. Sp.
Sh. Log. Vol.
Pv. Log. Vol. 1
Pv. Log. Vol. 2
24
26
28
30

Fig. 2

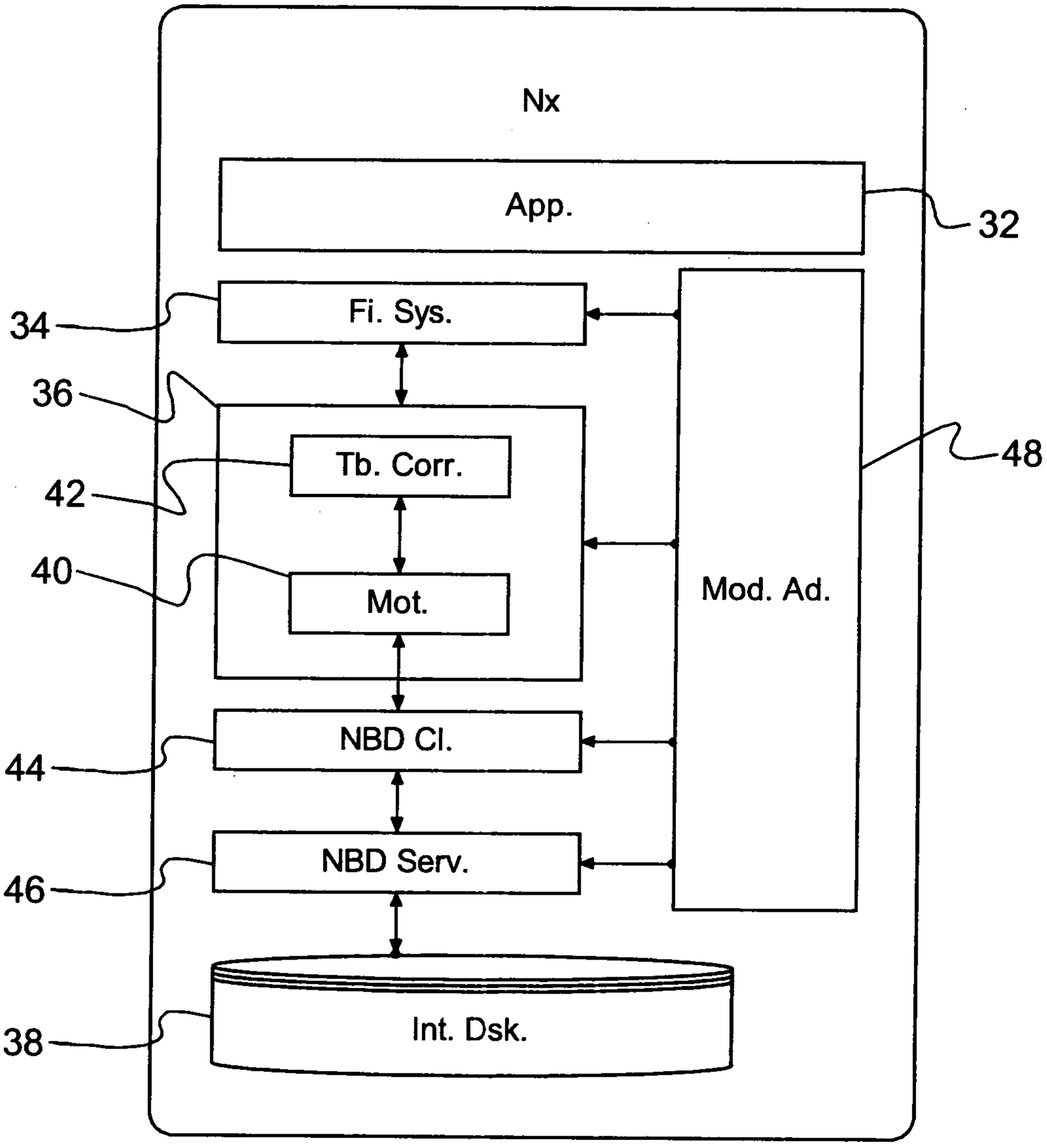

Nx
App.
32
Fi. Sys.
34
36
Tb. Corr.
42
40
Mot.
48
Mod. Ad.
NBD Cl.
44
NBD Serv.
46
Int. Dsk.
38

Fig. 3

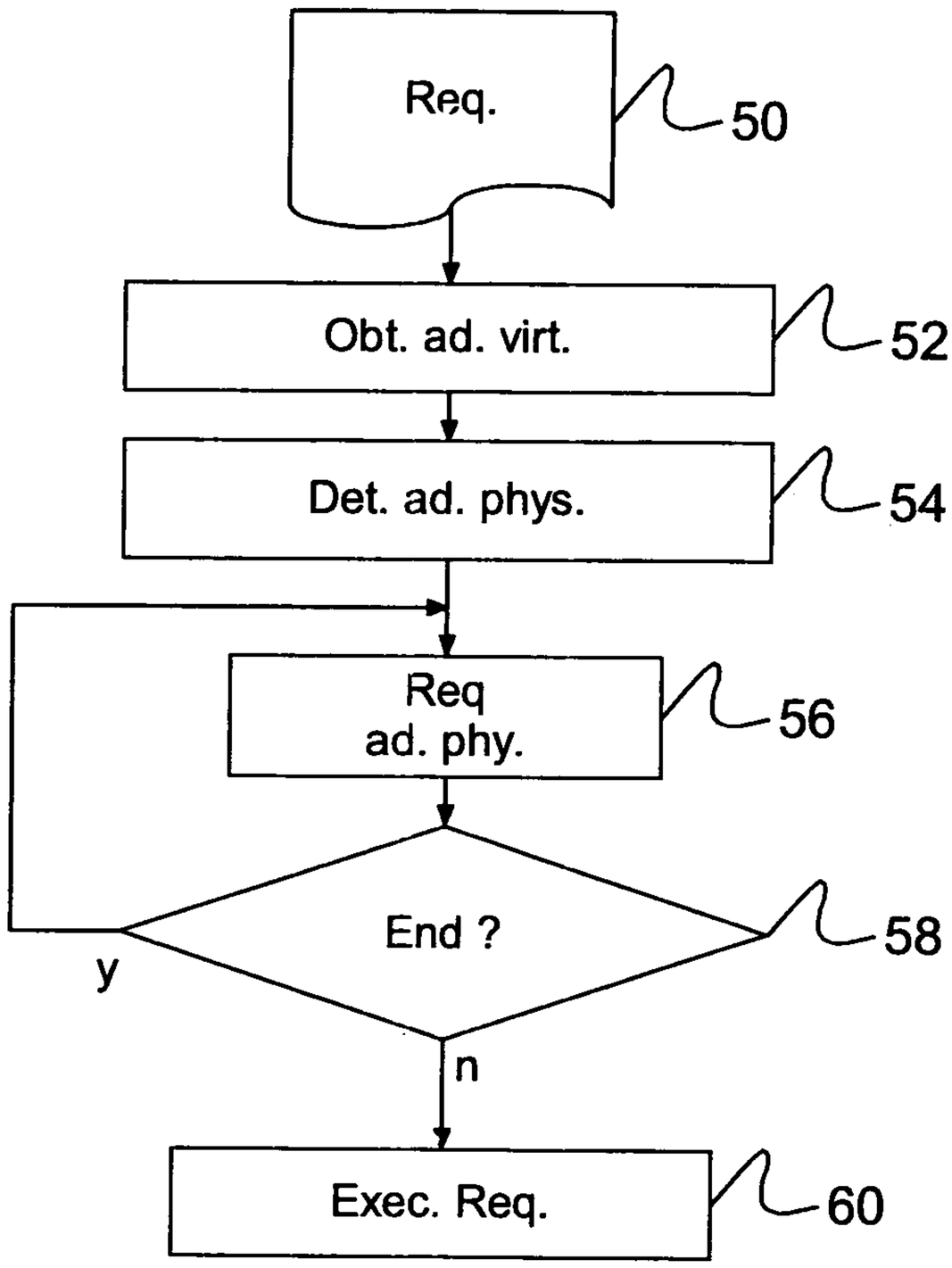

Req.
50
Obt. ad. virt.
52
Det. ad. phys.
54
Req
ad. phy.
56
End ?
58
y
n
Exec. Req.
60

Fig. 4

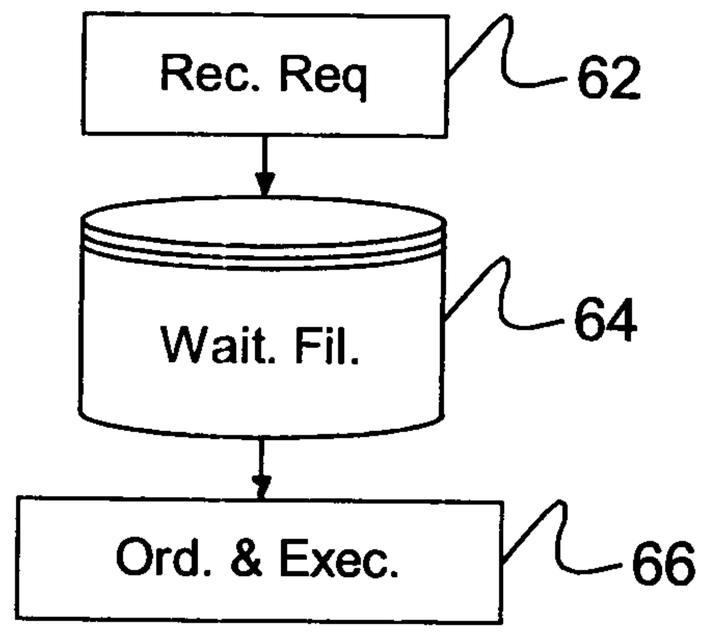

Rec. Req
62
Wait. Fil.
64
Ord. & Exec.
66

Fig. 5

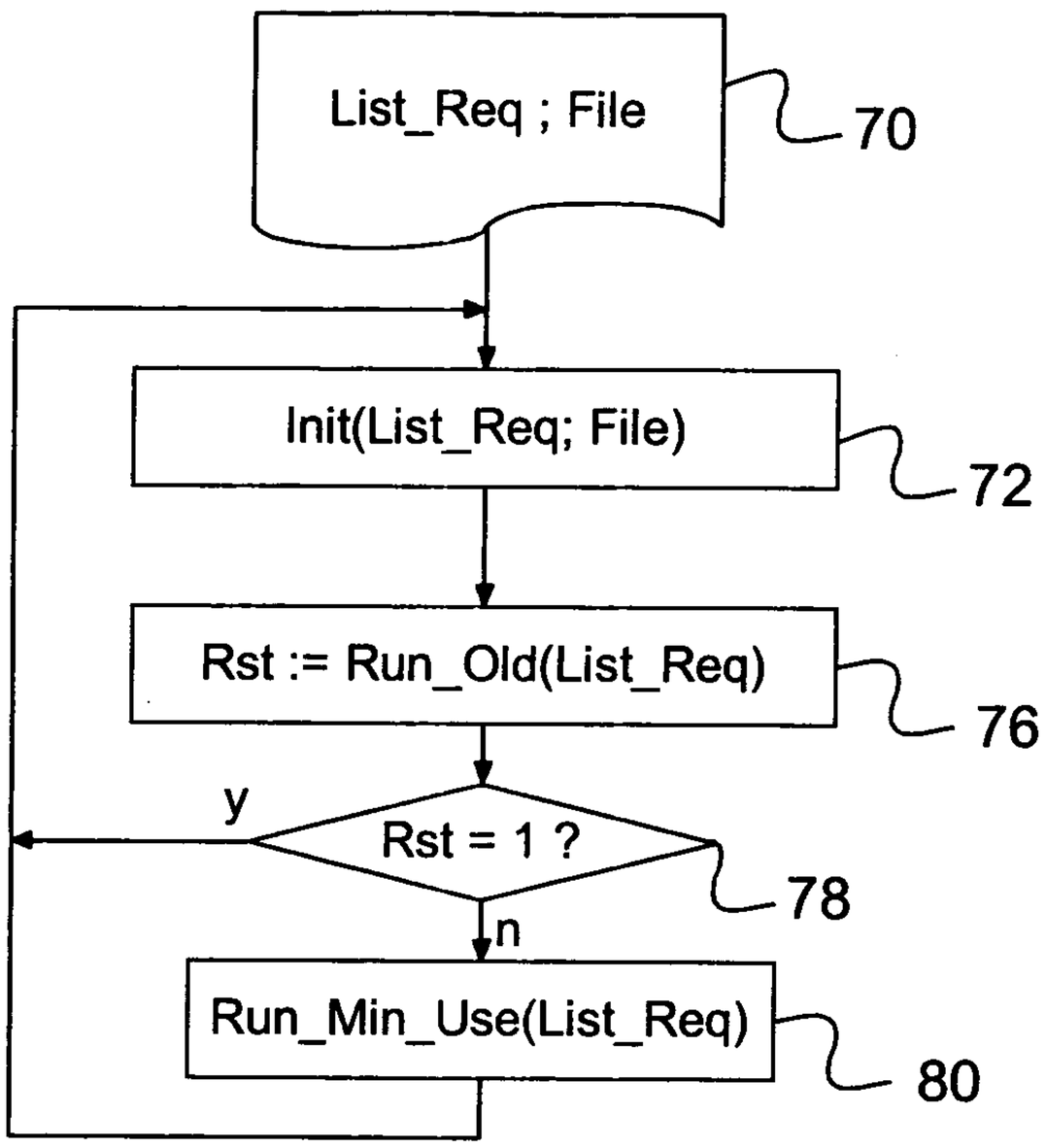

List_Req ; File
70
Init(List_Req; File)
72
Rst := Run_Old(List_Req)
76
y
Rst = 1 ?
78
n
Run_Min_Use(List_Req)
80

Fig. 6

716
Begin
700
Stat_Wt := 1
Wait(1)
714
Add_New_Req(List_Req; File)
702
y
2 Run. Req. ?
n
Stat_Wt ?
704
706
n
y
Stat_Wt := 0
708
y
List_Req = NULL ?
710
n
End
712

Fig. 7

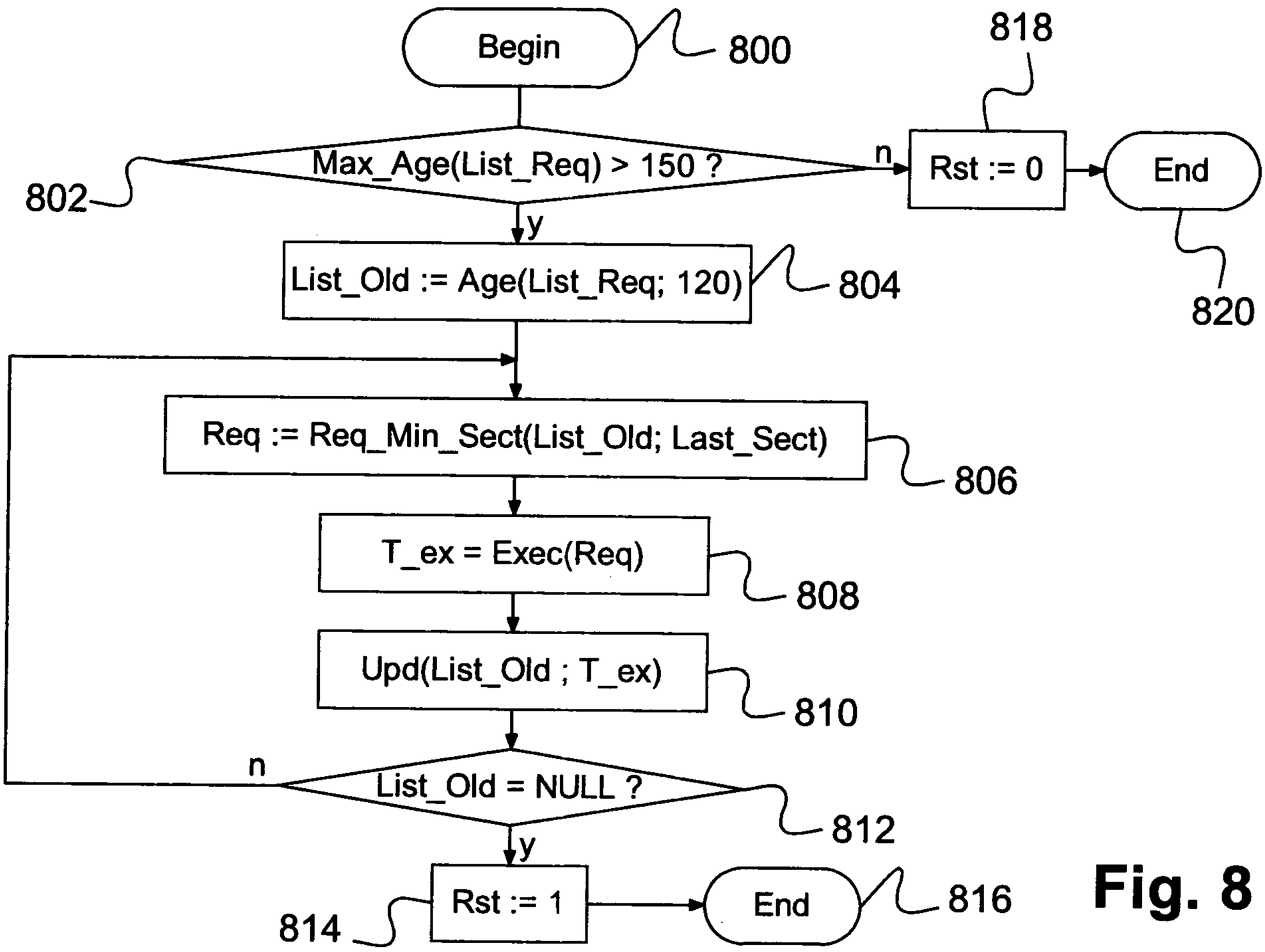

Begin
800
818
Max_Age(List_Req) > 150 ?
802
n
Rst := 0
End
820
y
List_Old := Age(List_Req; 120)
804
Req := Req_Min_Sect(List_Old; Last_Sect)
806
T_ex = Exec(Req)
808
Upd(List_Old ; T_ex)
810
n
List_Old = NULL ?
812
y
Rst := 1
814
End
816

Fig. 8

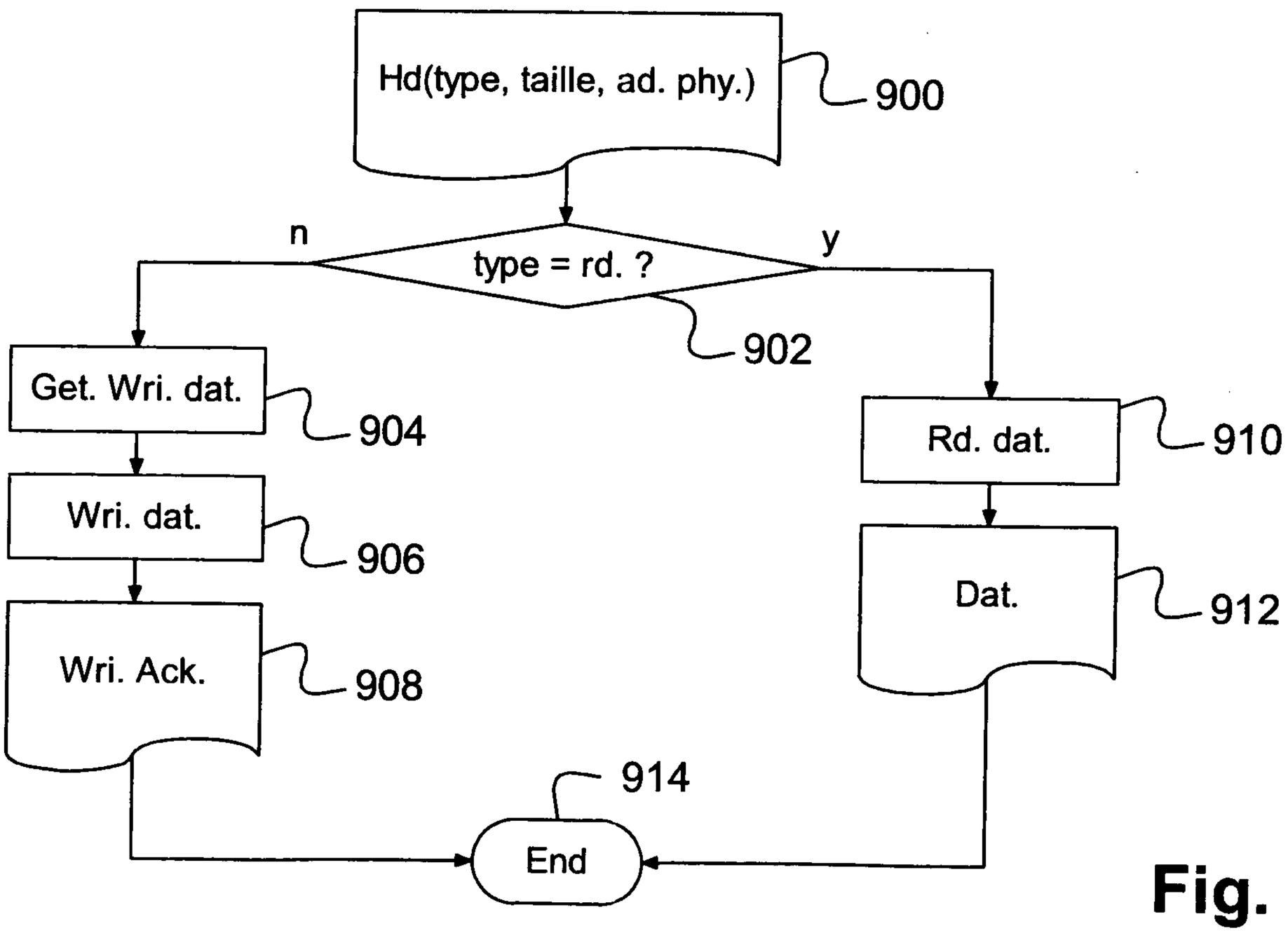

Hd(type, taille, ad. phy.)
900
n
type = rd. ?
902
y
Get. Wri. dat.
904
Wri. dat.
906
Wri. Ack.
908
Rd. dat.
910
Dat.
912
914
End

Fig. 9

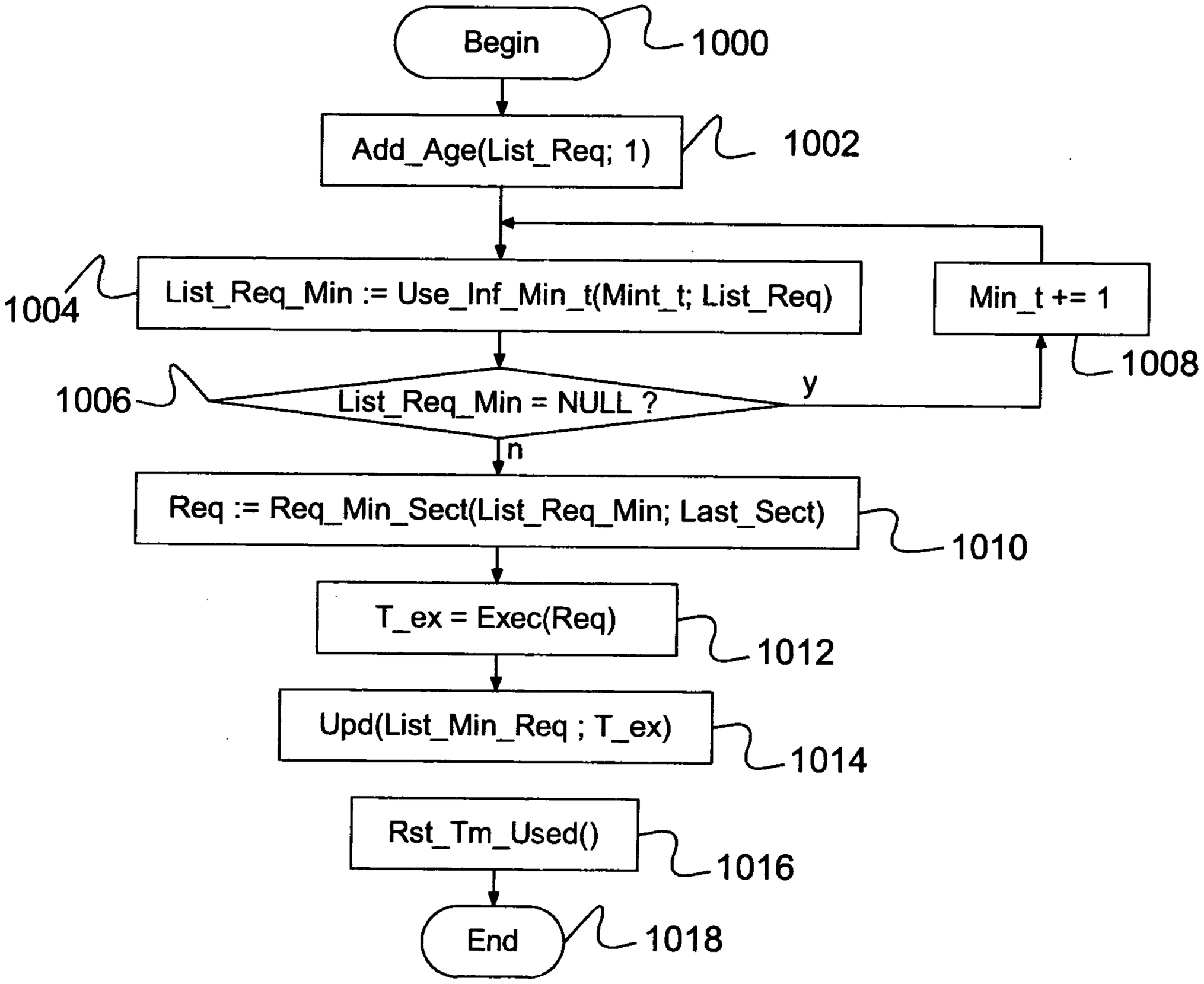

Begin
1000
Add_Age(List_Req; 1)
1002
1004
List_Req_Min := Use_Inf_Min_t(Mint_t; List_Req)
Min_t += 1
1008
1006
List_Req_Min = NULL ?
y
n
Req := Req_Min_Sect(List_Req_Min; Last_Sect)
1010
T_ex = Exec(Req)
1012
Upd(List_Min_Req ; T_ex)
1014
Rst_Tm_Used()
1016
End
1018

Fig. 10

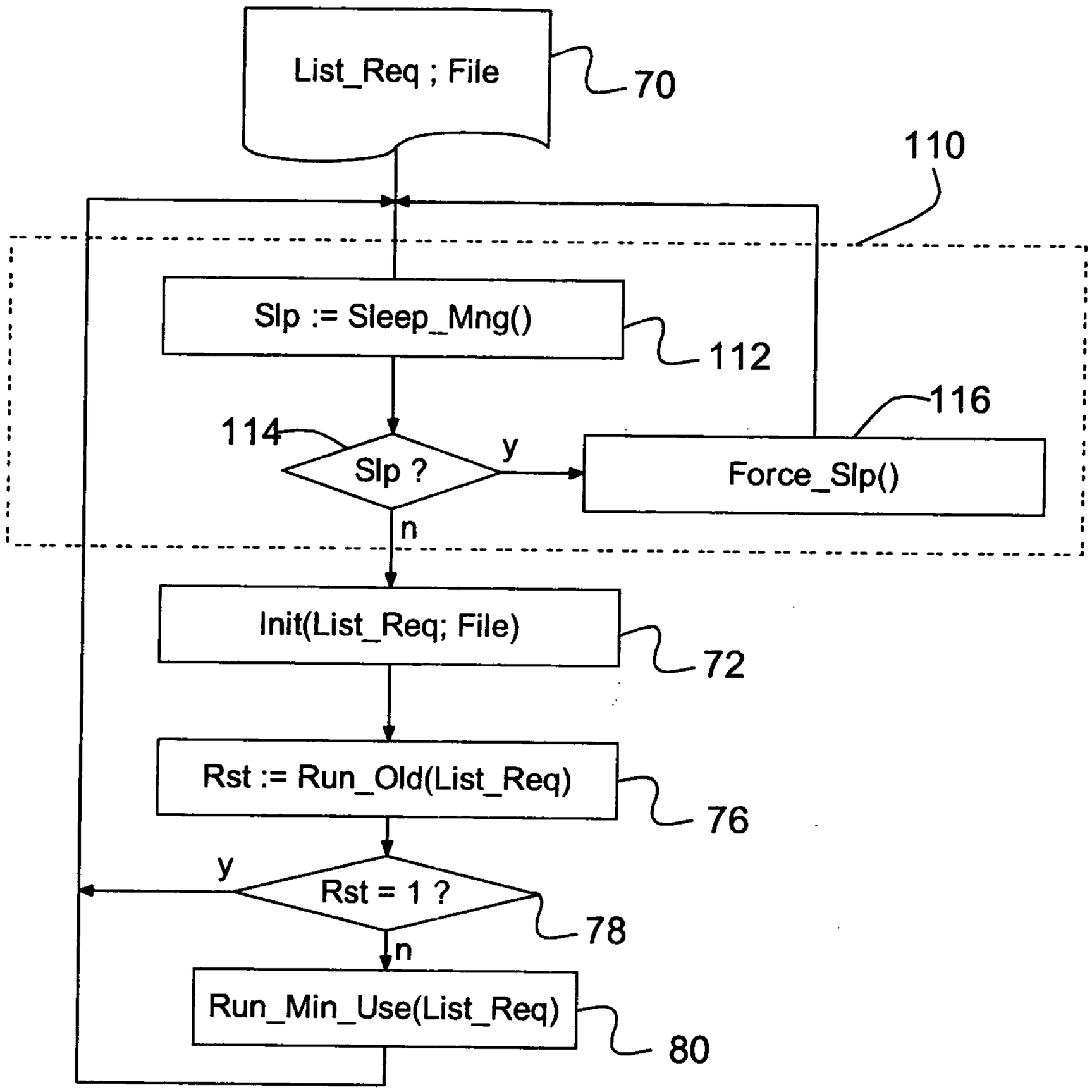

List_Req ; File
70
110
Slp := Sleep_Mng()
112
114
Slp ?
y
116
Force_Slp()
n
Init(List_Req; File)
72
Rst := Run_Old(List_Req)
76
y
Rst = 1 ?
78
n
Run_Min_Use(List_Req)
80

Fig. 11

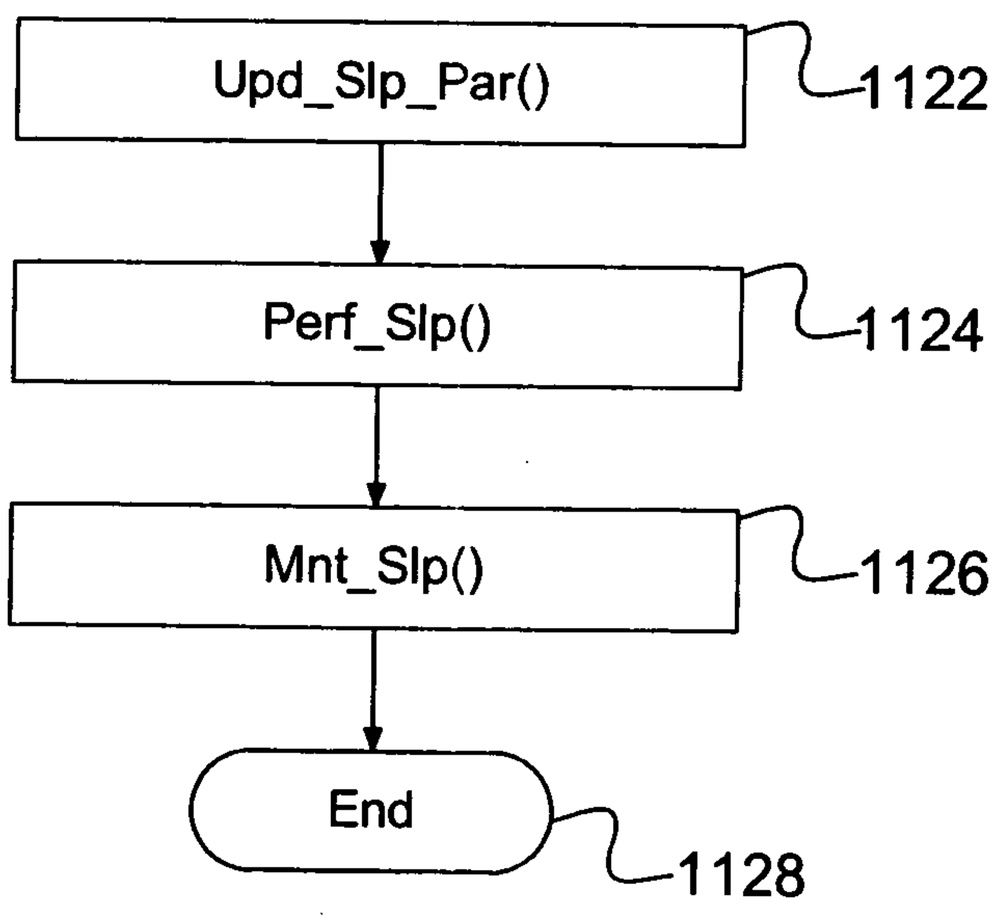

Upd_Slp_Par()
1122
Perf_Slp()
1124
Mnt_Slp()
1126
End
1128

Fig. 12

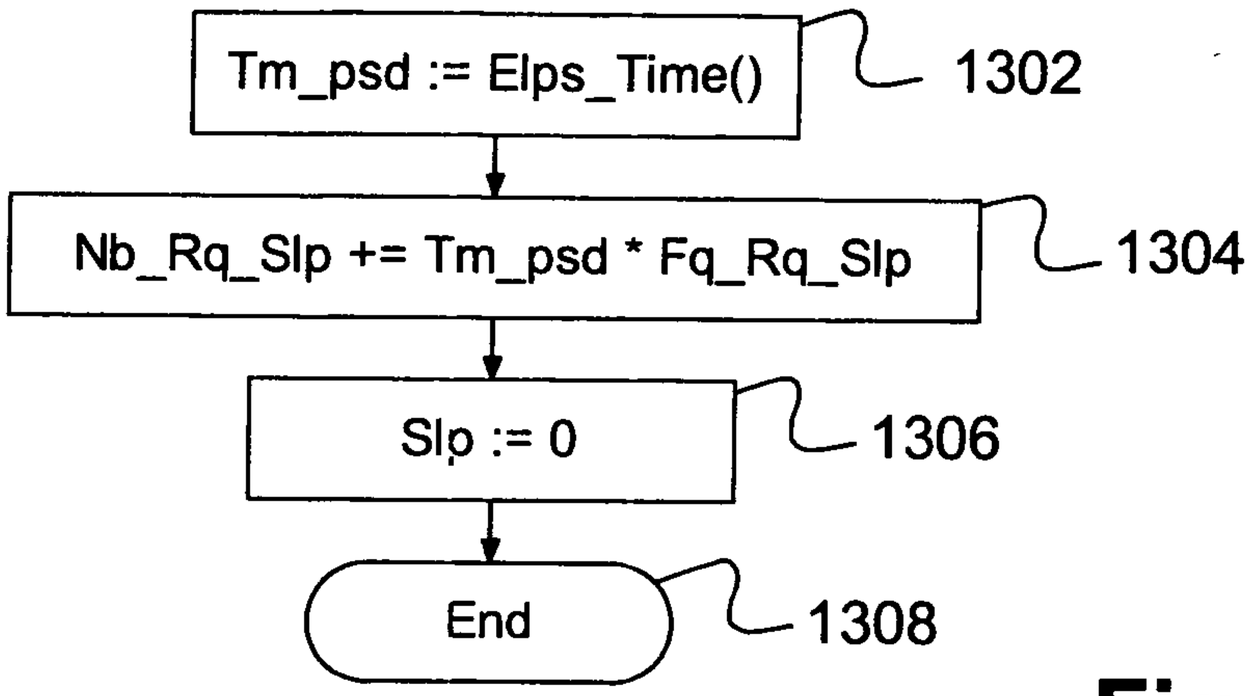


Fig. 13

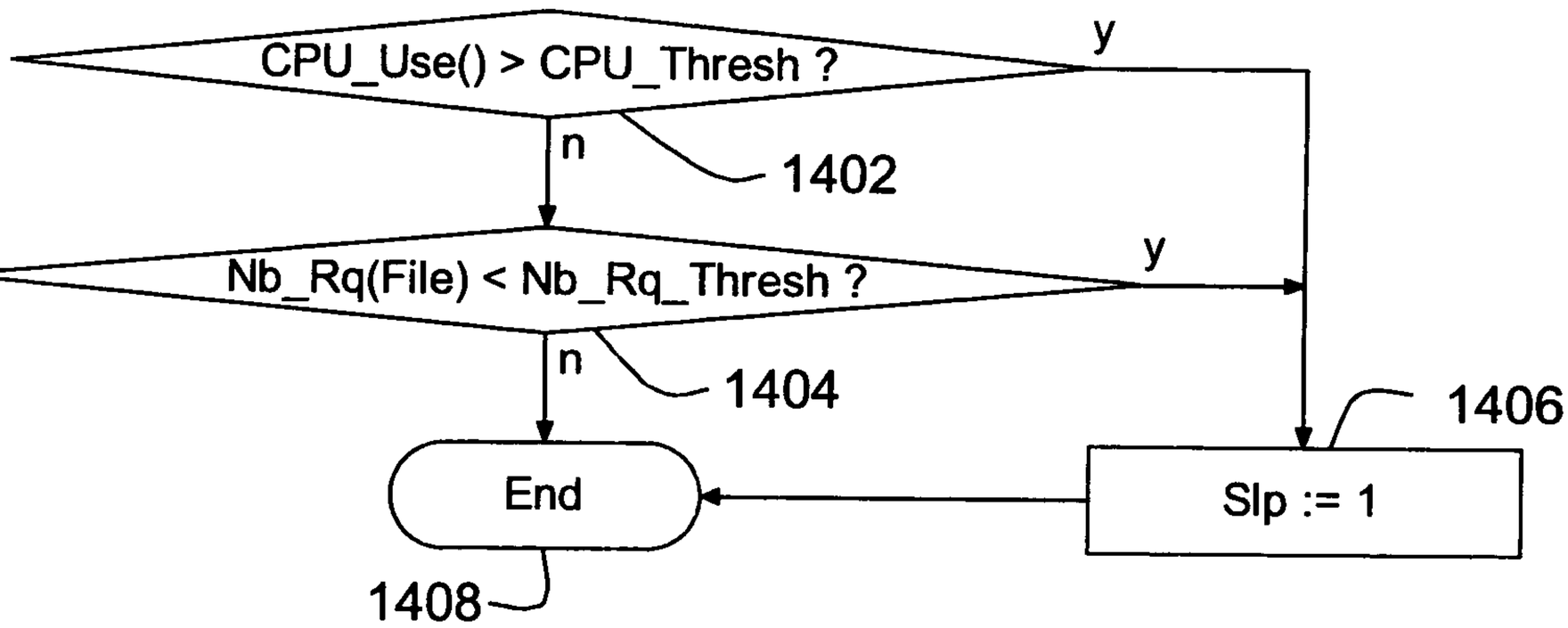


Fig. 14

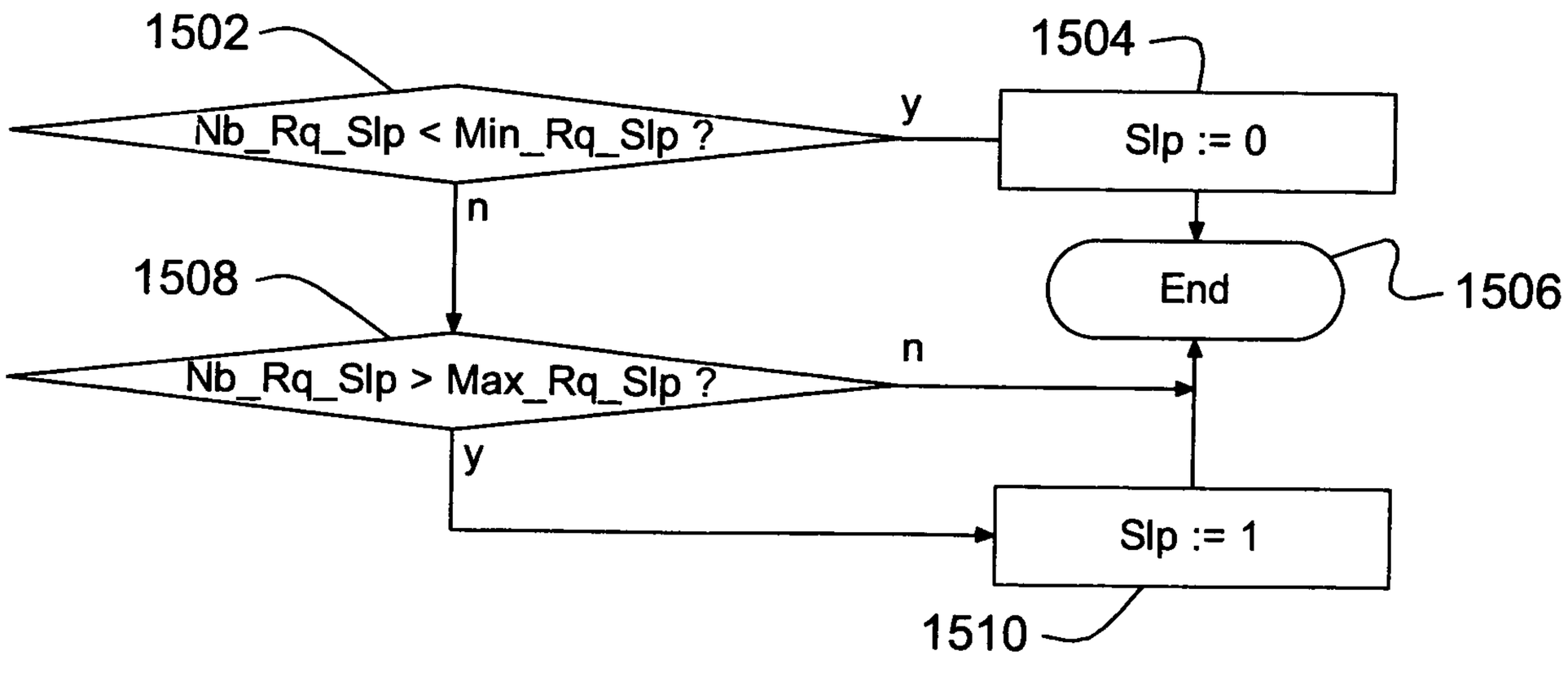
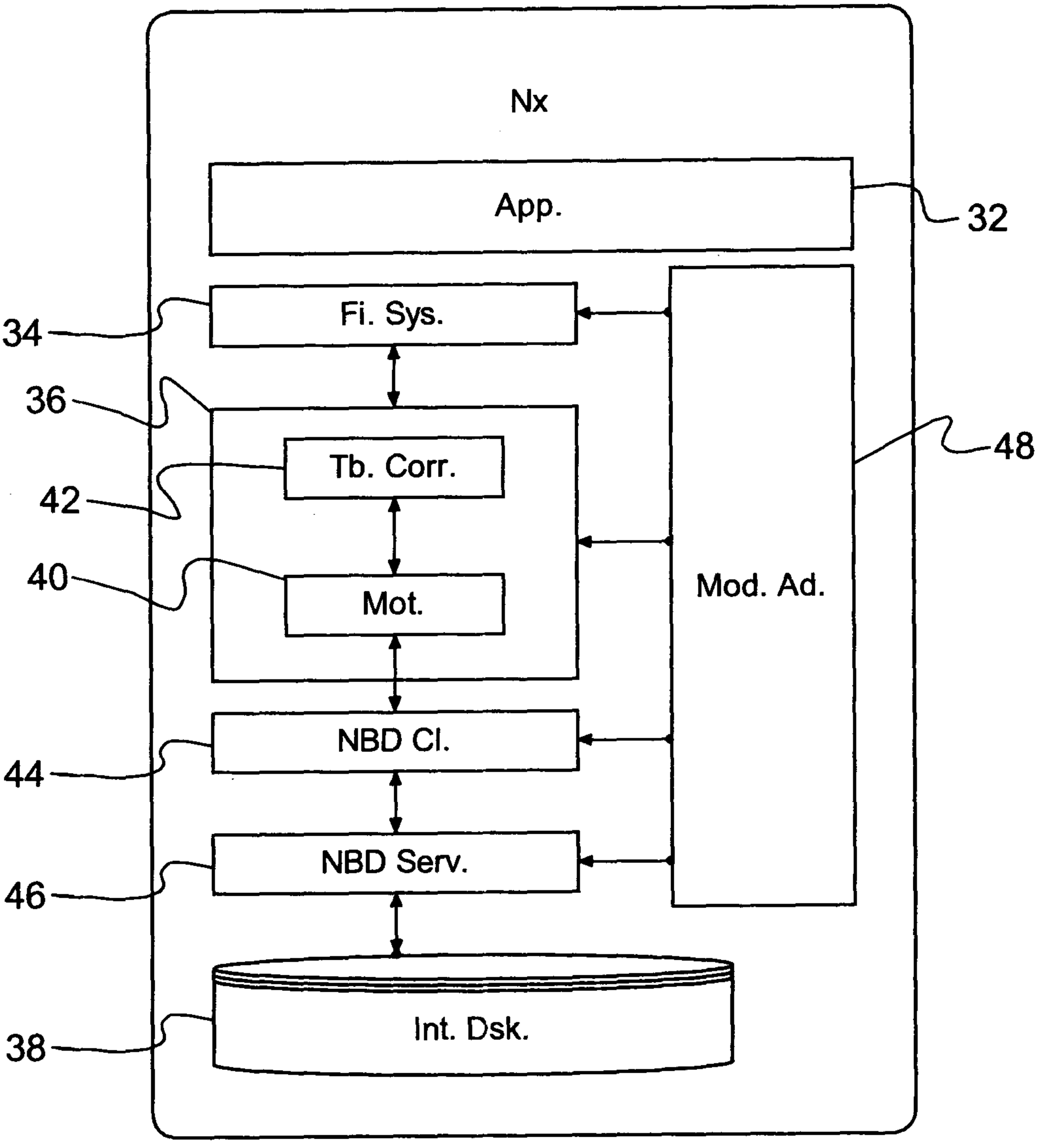


Fig. 15


Nx
App.
32
Fi. Sys.
34
36
Tb. Corr.
42
40
Mot.
Mod. Ad.
48
NBD Cl.
44
NBD Serv.
46
Int. Dsk.
38