A distributed algorithm to locally weaken the effects inherent in the CAP/Brewer's theorem when reading data on a distributed computer system

Jean-Philippe Fauvelle

How to locally weaken the effects inherent in the CAP/Brewer's theorem when reading data on a distributed computer system

Jean-Philippe Fauvelle

Contact : https://www.linkedin.com/in/jpfauvelle/

Abstract: The CAP (Consistency, Availability, Partition tolerance) also known as Brewer's theorem, states that it is impossible, on a distributed computer system, to guarantee at the same time the three qualities mentioned above. This issue affects all existing distributed engines and systems, which forces the administrators of these engines and systems to favor two qualities to the detriment of a third, when possible, despite the consequences. This paper technically and formally describes an inventive algorithmic process to locally weaken the said theorem when reading the data. This process combines three elements: a real-time system of key-value pairs distributed in memory cloud; a multi-planar index with concurrency avoidance; a distributed consolidated reading method called "LC4" based on a hysterical retroactive probabilistic effect. This inventive algorithmic process was implemented and tested successfully. Full understanding of this paper requires expertise in computer science, software engineering and algorithmics.

Keywords: Computer science, CAP theorem, Brewer's theorem, Distributed computing, Concurrent computing, Consistency, Availability, Partition tolerance.

SOMMAIRE

LISTE DES FIGURES

LISTE DES TABLEAUX

GLOSSAIRE

Terme	Synonymes	Signification
DAB		Automate de type Distributeur Automatique de Billets.
Équipement	Automate / ATM Client (informatique) Poste client	Équipement informatique, équipé d'un « progiciel client », à partir duquel un utilisateur consomme un service. Par exemple, un automate DAB / GAB est un équipement.
Flotte	Flotte d'automates	Ensemble des automates en production.
GAB		Automate de type Guichet Automatique de Banque, qui permet des opérations sur comptes en plus des retraits d'argent.
IMDBMS		*In-memory database management system.* Base de données en mémoire.
KVS		*Key Value Store.* Système de stockage de données sous forme de paires (clé, valeur).
Parc	Parc de serveurs	Ensemble des serveurs en production ayant pour fonction de traiter les requêtes provenant de la flotte d'automates.
Progiciel client	Progiciel automate	Progiciel installé sur chaque client.
Progiciel serveur		Progiciel installé sur chaque serveur.
RAM-Cloud		Système de données distribuées dans un nuage en mémoire.
Serveur	Serveur de traitement Serveur informatique	Serveur informatique, équipé d'un « progiciel serveur », traitant les requêtes informatiques provenant des équipements.
Succès de lecture d'une donnée		Lors du procédé de lecture d'une donnée : • Le succès de la lecture est dit **positif** lorsque le procédé a permis de lire la donnée existante. • Le succès de la lecture est dit **négatif** lorsque le procédé n'a pas permis de lire la donnée et n'a pas non plus rencontré d'erreurs techniques. • Le succès de la lecture est dit en **erreur** lorsque le procédé n'a pas permis de lire la donnée en raison d'une ou plusieurs erreurs techniques.
Utilisateur	Individu Porteur de carte	Une personne qui utilise un équipement par le biais du progiciel client installé sur ledit équipement.

Tableau 1 – Glossaire.

1 DESCRIPTION TECHNIQUE

1.1 DESCRIPTION DE LA PROBLÉMATIQUE

Une application quelconque nommée « l'application », utilisatrice de l'invention, est instanciée horizontalement sur une plateforme matérielle quelconque constituée de **N** nœuds informatiques. L'application peut éventuellement être multi-instanciée verticalement sur chaque nœud.

L'application requiert d'accéder à des données présentes dans un moteur de données.

Ledit moteur, ci-après désigné par « le moteur de données » ou « le composant inventé » ou « l'invention », se conforme à l'ensemble des exigences ci-après :

Ref.	Exigence
EX1	L'application peut lire, écrire et supprimer des données dans le moteur de données.
EX2	L'application peut accéder à chaque donnée présente dans le moteur de données à partir d'une clé primaire.
EX3	Chaque donnée comprend un ensemble quelconque d'informations, structurées ou non.
EX4	Les données peuvent être regroupées en tables indexées, accédées sans connaitre les clés primaires.
EX5	L'accès à partir de sa clé primaire, à toute donnée de taille inférieure à 1Ko, s'exécute en un temps moyen : • D'environ 100 µs sur un serveur matériel standard (exemple : Hewlett-Packard DL380 ou similaire). • D'environ 10 µs, voire moins, lorsque les serveurs sont associés soit aux technologies Infiniband plus RDMA (*Remote Direct memory Acces*) soit aux technologies HSE (*High Speed Ethernet*) plus RoCE (*RDMA over Converged Ethernet*).
EX6	L'accès à toute donnée à partir de sa clé primaire, s'effectue avec une complexité temporelle en **O(1)**.
EX7	L'application manipule un volume de données quelconque non limité par l'espace d'adressage 64bits.
EX8	L'application requiert une résilience nominale des données même en présence de doubles pannes.
EX9	L'application requiert que le moteur implémente un mécanisme de détection et correction automatiques des incohérences et pertes de données provoquées par des dysfonctionnements. En particulier, lorsqu'un dysfonctionnement a empêché la mise à jour ou la création de certaines données, et même lorsque ce dysfonctionnement est révolu, le moteur de données est capable : • De détecter qu'une donnée manquante l'est à tort (faux négatif). • De détecter qu'une donnée présente est obsolète (incohérence positive). • De réparer les données correspondant aux 2 cas précités.
EX10	La plateforme hébergeant le moteur de données peut être redimensionnée (ajout/retrait de nœuds) sans interruption de service ni impact sur la résilience des traitements et la résilience des données.
EX11	Le moteur de données ne requiert pas d'installer ni d'administrer un complexe logiciel de type NoSQL ou SQL.
EX12	Le moteur de données ne requiert pas de flux de données inter-nœuds.
EX13	Le moteur de données possède une capacité de mise à l'échelle horizontale.

Tableau 2 – Exigences.

1.2 DESCRIPTION DE L'INVENTION

1.2.1 PRINCIPE

L'application utilise le moteur de données inventé.

L'invention est implémentée sous la forme d'un composant logiciel, plus précisément d'une librairie logicielle embarquée dans chaque instance de l'application.

L'invention utilise des moteurs de type KVS (*Key Value Store*) hébergés sur plusieurs nœuds, lesdits moteurs KVS prenant en charge le stockage physique des données dans la mémoire desdits nœuds.

Ces moteurs KVS sont des services standards gérés par le système d'exploitation et ne nécessitant pas d'administration. À titre d'exemple, le KVS Memcached open-source est un simple package qui s'installe en quelques minutes sur des nœuds de type Linux Redhat, et qui nécessite seulement de spécifier un dimensionnement mémoire alloué au service KVS.

Les services KVS répondent uniquement aux exigences EX1, EX2, EX5, EX6 [Tableau 2].

L'invention, lorsqu'elle est associée à un KVS, répond à l'ensemble des exigences [Tableau 2] et ne requiert pas d'installer un complexe moteur de données de type NoSQL ou SQL puisque toute la logique de fonctionnement d'un moteur de données est implémentée dans le composant inventé, lui-même embarqué sous forme d'une librairie dans chaque instance de l'application utilisatrice.

En d'autres termes, une application quelconque utilise l'invention sous la forme d'un composant embarqué, qui à son tour utilise en client/serveur des moteurs KVS standards.

Dans le cas général, les instances de l'application et les services KVS peuvent être hébergés sur des nœuds distincts [Figure 1].

Dans le cas optimal, les instances de l'application et les services KVS sont hébergés sur **N** nœuds communs afin de bénéficier du principe de localité entre traitements et données inhérent au paradigme big-data [Figure 2].

Les services KVS n'inter-opèrent pas et n'intègrent aucun mécanisme de résilience ou de cohérence.

Chaque donnée est matérialisée sous la forme d'une paire {clé ; donnée} stockée dans les services KVS et ainsi formée :

```
{    clé = (type_paire, nom_table, clé_primaire) ;
  donnée = (objet, horodatage) }
```

Où :

- type_paire est une constante, arbitrairement « table », indiquant que la paire contient les informations d'une table de données, par opposition aux informations relatives à un index ou à un autre type.
- nom_table est le nom « métier » attribué à la table par l'application qui utilise l'invention. Ce nom intégré dans la clé de la paire, permet d'éviter les collisions entre les paires de tables différentes.
- clé_primaire est la clé « métier » utilisée pour identifier la donnée de la paire de façon unique dans la table nom_table du point de vue de l'application utilisatrice de l'invention.

La donnée de la paire contient un objet quelconque fourni par l'application, puis sérialisé et éventuellement compressé / chiffré à la volée par l'invention lors de l'écriture de la paire, les actions inverses étant réalisées à la volée par l'invention lors de la lecture de la paire.

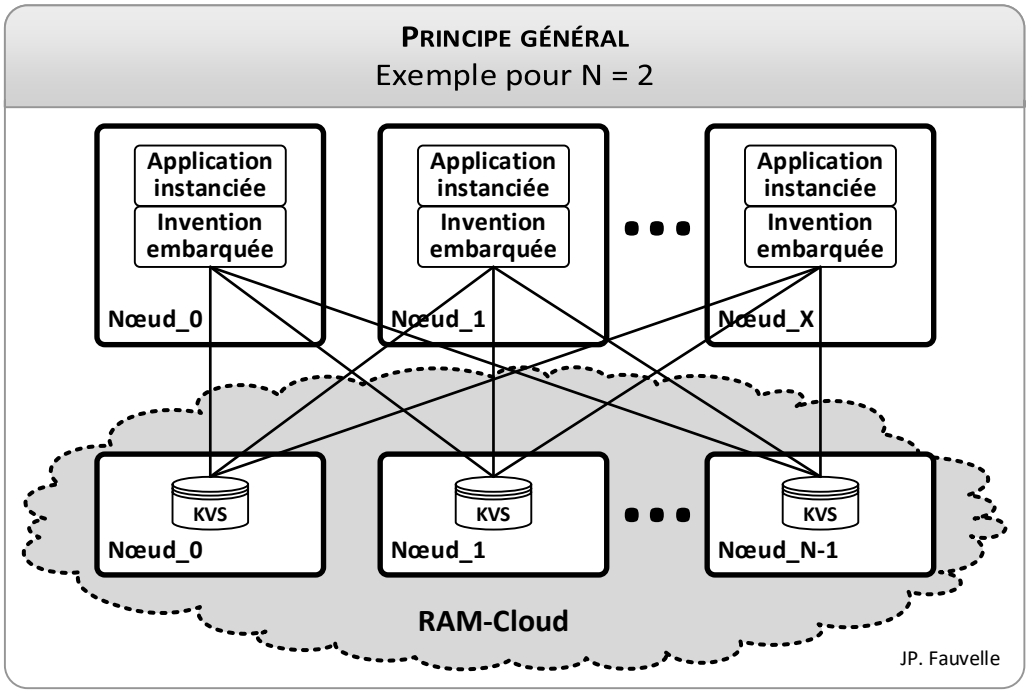

Figure 1 – Principe général.

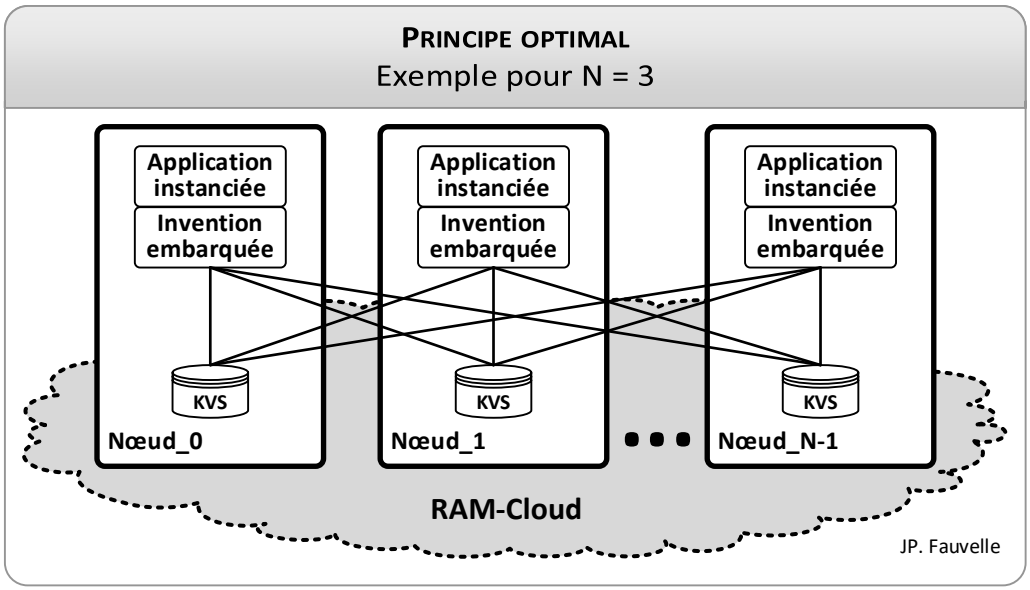

Figure 2 – Principe optimal.

1.2.2 INDEX MULTIPLANAIRE

1.2.2.1 BESOIN

Pour rappel, une paire appartenant à une table de données est ainsi constituée :

```
{    clé = (type_paire, nom_table, clé_primaire) ;
  donnée = (objet, horodatage) }
```

Via l'invention, l'application peut toujours accéder directement à une paire à partir de sa clé primaire, puisque la clé composite de ladite paire se déduit de sa clé primaire. À contrario, l'application ne peut pas accéder directement à une paire si elle n'en connait pas la clé primaire.

Pour cette raison, l'invention est capable d'opérer un index par table afin de permettre à l'application d'accéder indirectement aux données indexées de ladite table sans connaitre leurs clés primaires [Figure 3].

Par défaut, aucune donnée n'est indexée. Lorsque l'application souhaite qu'une donnée soit indexée, elle demande explicitement à l'invention de procéder à l'indexation de ladite donnée. L'index d'une table recense tout ou partie des clés qui existent dans ladite table. Aux extrêmes, l'index est vide ou contient la totalité des clés.

L'index d'une table est créé à la volée lors de la première demande d'indexation portant sur une donnée de ladite table.

1.2.2.2 ARCHITECTURE

L'index d'une table est découpé en **F** fragments, une constante configurable [301].

Chaque fragment est implémenté sous la forme d'une paire ainsi constituée :

```
{    clé_index = (type_paire, nom_table, num_frag, hôte_client) ;
  donnée_index = (liste_clés, horodatage) }
```

Où :

- clé_index est une chaîne de caractères contenant les informations (type_paire, nom_table, num_frag, hôte_client), où :
    - type_paire est une constante, arbitrairement « index », indiquant que la paire contient les informations d'un index, par opposition aux informations relatives à une table ou à un autre type.
    - nom_table est le nom « métier » attribué à la table par l'application qui utilise l'invention, cette table faisant l'objet de l'indexation.
    - num_frag est le numéro du fragment de l'index, compris dans l'intervalle [1, **F**].
    - hôte_client est le nom d'hôte (*hostname*) depuis lequel l'application utilisatrice de l'invention effectue la requête vers le RAM-Cloud, l'application étant cliente du RAM-Cloud.
- donnée_index contient la liste liste_clés de toutes les clés primaires X des paires contenues dans la table nom_table, ayant fait l'objet d'une indexation, et pour lesquelles hachage(X) = num_frag.

La présence de l'information hôte_client dans clé_index a pour effet que chaque nœud de l'application utilisant l'invention possède et gère son propre **plan de l'index**, isolé des autres plans gérés par les autres nœuds [302].

Le découpage en **F** fragments répond à une double problématique : respecter l'éventuelle limitation de taille d'une paire dans le produit KVS utilisé, et augmenter les performances en ne manipulant que des sous-ensembles de clés.

1.2.2.3 INDEXATION D'UNE PAIRE

Pour rappel, une paire appartenant à une table de données est ainsi constituée :

```
{    clé = (type_paire, nom_table, clé_primaire) ;
  donnée = (objet, horodatage) }
```

Pour rappel, une paire constituant un fragment d'index de la table est ainsi constituée :

```
{    clé_index = (type_paire, nom_table, num_frag, hôte_client) ;
  donnée_index = (liste_clés, horodatage) }
```

L'indexation d'une paire appartenant à la table nom_table, consiste à ajouter la clé primaire de ladite paire dans un fragment de l'index de ladite table.

Le fragment est déterminé par hachage uniforme sur cette clé primaire, ce dont il découle d'une part que les différents fragments de l'index sont statistiquement de même taille, et d'autre part que le fragment d'appartenance est déterministe – ce qui autorise une indexation rapide.

L'ajout d'une clé_primaire dans l'index d'une table nom_table consiste :

1. à déterminer num_frag = hachage(clé_primaire).
2. puis, de façon atomique :
   a. à lire la paire associée au fragment num_frag pour en extraire la liste des clés primaires déjà indexées, affectées à ce fragment [310].

      Cette paire est accessible via sa clé_index = (type_paire, nom_table, num_frag, hôte_client).

      La lecture est effectuée sur les **R** réplicas de la paire. En cas d'incohérence, l'horodatage le plus récent l'emporte.
   b. à ajouter la clé_primaire dans la liste si elle n'y figure pas déjà.
   c. à réécrire la paire [311].

      L'écriture est effectuée sur les **R** réplicas de la paire.

L'atomicité des opérations d'indexation ci-dessus est assurée par une exclusion mutuelle portant sur le triplet (nom_table, num_frag, hôte_client). Cette exclusion mutuelle est implémentée par un verrou portant sur le couple (nom_table, num_frag) et localisé sur le nœud hôte_client de l'application [312].

Ledit verrou est positionné par l'invention sur le nœud depuis lequel l'application utilisatrice effectue la requête d'indexation, car l'invention est embarquée dans chaque instance de l'application et a donc accès au nœud de l'application.

La seule contention existante ne se produit qu'entre plusieurs instances de l'application, qui s'exécutent simultanément sur un même nœud, et qui accèdent simultanément en écriture (demande d'indexation d'une clé) sur un même fragment (une paire) de l'index d'une même table [313].

Ce procédé d'index multiplanaire est avantageux : différentes instances de l'application, s'exécutant sur des nœuds distincts, ne souffrent d'aucune concurrence pour écrire dans un même index, y compris pour écrire dans un même fragment. En d'autres termes, ce procédé évite la concurrence inter-nœuds lors de l'écriture nécessaire à l'indexation d'une clé, ce qui est cohérent avec la meilleure performance et la capacité de mise à l'échelle horizontale.

1.2.2.4 LECTURE DES DONNÉES INDEXÉES

Lorsqu'une instance de l'application s'exécutant sur un nœud donné souhaite lire la totalité des données indexées, alors l'invention embarquée au sein de ladite instance [Figure 3] :

1. procède à la lecture, sans aucune contention, de toutes les paires de tous les fragments pour tous les plans existant [320].
2. fusionne et dé-doublonne les clés indexées.
3. vérifie l'existence réelle, dans la table des données, des paires dont les clés sont indexées. [321].
4. retourne à l'application le résultat constitué des paires indexées dont l'existence est vérifiée.

Ce procédé d'index multiplanaire est avantageux : il évite la concurrence inter-nœuds lors de la lecture des données indexées, ce qui est cohérent avec la meilleure performance et la capacité de mise à l'échelle horizontale.

1.2.2.5 COMPACTAGE D'INDEX

Rien n'interdit qu'une donnée non existante dans une table soit indexée ou qu'une donnée indexée soit ultérieurement supprimée d'une table. Il en découle que les index peuvent contenir des références sur des clés qui ne sont pas ou plus présentes dans une table.

Afin d'éviter l'accumulation de telles obsolescences et de réduire l'empreinte mémoire inhérente, l'invention fournit à l'application un mécanisme technique qui assure le nettoyage des index, automatiquement et de façon transparente.

Cette fonction de nettoyage s'exécute en plusieurs fois, et chaque itération s'exécute en durée maximum garantie spécifié par l'application.

Ce procédé est avantageux : il permet à l'application de fonctionner en durée courte et garantie c.-à-d. en temps réel.

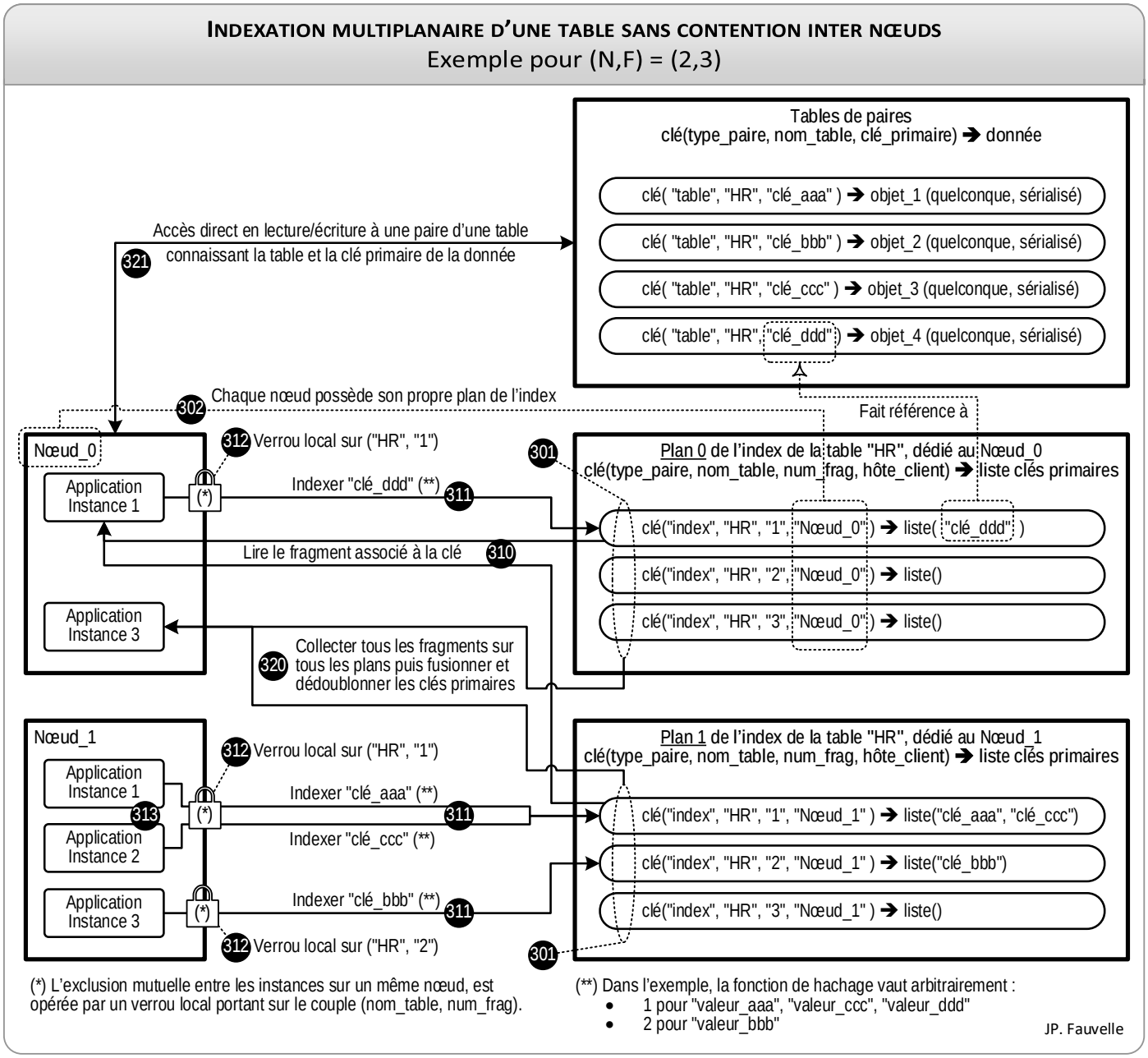

Figure 3 – Indexation multiplanaire d'une table sans contention inter nœuds.

1.2.3 UNIFORMITÉ, CONSISTANCE ET DÉTERMINISME DU HACHAGE DES RÉPLICAS

Qu'elle appartienne à une table ou à un index, toute donnée (v) associée à une clé (k), se matérialise sous la forme d'une paire ainsi constituée :

```
{ clé k ; donnée v = (objet x, horodatage h) }
```

Ladite paire est publiée sous la forme de **R** réplicas identiques, localisés sur **R** nœuds KVS distincts parmi les **N** existants.

La localisation du premier réplica d'une paire se calcule par hachage sur sa clé (k), les autres réplicas étant localisés sur les nœuds qui suivent dans l'ordre de la liste complète des nœuds définie par configuration.

L'algorithme de hachage est intrinsèquement **déterministe** : la connaissance de la clé (k) détermine la liste des **R** nœuds contenant les réplicas, si ladite donnée existe.

L'algorithme de hachage est **uniforme** : les données sont uniformément distribuées sur les KVS des **N** nœuds. Il en découle que la probabilité pour un nœud d'héberger l'un des réplicas d'une donnée, vaut **R/N**.

L'algorithme de hachage est **consistant** : en cas de redimensionnement de la plateforme (modification haussière ou baissière du nombre de nœuds), le nombre de paires subissant un décalage est limité à **K/N** paires où **K** est le nombre total de clés.

À titre d'exemple, une fonction *Hash* implémentant un hachage uniforme et raisonnablement consistant, qui, à une clé (k), associe une valeur $Hash(k) \in [0, N[$ pourrait être :

$$Hash(k) = \left\lfloor \frac{MD5(k)\ modulo\ 10^4}{10^4} * N \right\rfloor \ ;\ Hash(k) \in [0, N[\ ;\ N \ll 10^4$$

1.2.4 ÉCRITURE CONSOLIDÉE D'UNE PAIRE DANS LE RAM-CLOUD

Pour rappel, une paire associée à une clé (k) est ainsi constituée :

```
{ clé k ; donnée v = (objet x, horodatage h) }
```

L'écriture consolidée d'une paire consiste à écrire les **R** réplicas sur **R** instances KVS [Figure 4].

Cette action a pour effet de créer les nouvelles données ou de régénérer les données éventuellement manquantes suite à un incident ou un redimensionnement de la plateforme (*re-sharding*).

L'écriture consolidée d'une paire est un succès si le nombre de réplicas écrits sans erreur atteint le quorum **Q** (majorité absolue des réplicas) puisque cela garantit l'atteinte du même quorum lors d'une lecture ultérieure de ladite paire.

L'invention réalise cette opération comme suit :

1. Calculer par hachage déterministe sur la clé (k), les **R** nœuds devant recevoir les **R** réplicas.
2. Insérer dans l'objet à écrire une version de type horodatage en microsecondes, identique pour les **R** réplicas.
3. Itérativement, écrire les **R** réplicas [401, 402, 403].

Par défaut, les réplicas sont écrits de manière synchrone [401, 402]. Si le nœud de l'application invoquant l'écriture fait partie des nœuds contenant l'un des **R** réplicas, alors ce nœud local reçoit la première écriture synchrone [401].

Dès que le nombre d'écritures synchrones réussies atteint le quorum **Q**, alors l'écriture des réplicas restant s'effectue de manière asynchrone [403].

Du point de vue de l'application qui invoque l'écriture consolidée via le composant, les écritures asynchrones ne contribuent pas au temps de réponse. À titre d'exemple, pour **N** = 4 et **R** = 3, la durée d'écriture consolidée vue par l'application correspond à la durée de 2 écritures.

Ce procédé est avantageux : il permet d'atteindre la meilleure complexité en temps permise par la théorie, sans pénaliser la résilience, quels que soient le volume des données et le nombre de nœuds.

	Temps de réponse vu par l'application				
Complexité moyenne en temps dans les cas nominaux	**Écritures synchrones**			**Écritures asynchrones**	
	Quantité	**Probabilité**		**Quantité**	**Probabilité**
Écriture des R réplicas.	$Q = \left\lceil \frac{R}{2} \right\rceil$	1	+	$R - Q = R - \left\lceil \frac{R}{2} \right\rceil$	1

Tableau 3 – Complexité moyenne en temps des accès en écriture.

1.2.5 CONDITION TECHNIQUE, ÉTAT HYSTÉRÉTIQUE, RÉMANENCE D'UN SERVICE KVS

L'invention utilise trois caractéristiques [Figure 5] :

- La **condition technique** (ou **condition**) d'un nœud/service KVS [511].
- L'**état hystérétique** (ou **état**) d'un nœud/service KVS, calculé à partir de sa condition [510].
- La **période de rémanence** (ou **rémanence**) d'un nœud/service KVS, calculée à partir de son état et superposée audit état [560].

L'application utilisatrice peut être opérée sur plusieurs nœuds, chacun d'entre eux évalue lesdites caractéristiques indépendamment des autres. Autrement dit, des nœuds distincts opérant l'application utilisatrice peuvent à un instant donné évaluer différemment lesdites caractéristiques concernant un même nœud/service KVS.

Chaque nœud KVS possède une condition technique égale au statut {en erreur technique, non en erreur} de la plus récente requête en lecture ou écriture exécutée sur ledit nœud par l'application via l'invention [511].

Pour chaque nœud KVS, l'invention calcule un état hystérétique **«** disponible **»** ou « indisponible » via un compteur fonctionnant en cycle d'hystérésis par rapport à la condition technique [510].

À cette fin, pour chaque requête en lecture ou écriture effectuée par l'application utilisatrice opérée sur un nœud nœud_0 via l'invention, vers un nœud KVS localisé sur un nœud (s), l'invention observe le statut de ladite requête c.-à-d. la condition technique du nœud (s) vue par le nœud nœud_0 [511]. Selon que ce statut est ou non en erreur, un compteur d'erreurs (c_s) associé au nœud (s) et opéré sur le nœud nœud_0 est incrémenté ou décrémenté [510]. La valeur dudit compteur est bornée sur l'intervalle [0, **E**] où **E** est une constante configurable positive non nulle correspondant à un nombre maximum d'erreurs.

Ledit compteur est opéré sur le nœud nœud_0 comme suit :

- Lorsque la valeur du compteur (c_s) est incrémentée de E-1 à E, et si l'état hystérétique (e_s) est « disponible » [510a], alors l'état hystérétique (e_s) du nœud (s) reçoit la valeur « indisponible » et l'horodatage (i_s) correspondant à cette inversion d'état dudit nœud reçoit l'heure courante [510].
- Lorsque la valeur du compteur (c_s) est décrémentée de 1 à 0, et si l'état hystérétique (e_s) est « indisponible » [510b], alors l'état hystérétique (e_s) du nœud (s) reçoit la valeur « disponible » et l'horodatage (i_s) correspondant à cette inversion d'état dudit nœud reçoit l'heure courante [510].
- Dans les autres cas, les valeurs de (e_s) et (i_s) demeurent inchangées [510].

Chaque fois que l'état (e_s) est modifié, le triplet (s, e_s, i_s) est mémorisé 512 dans une table **TableEtats** 520 opérée sur le nœud nœud_0. Des nœuds distincts opérant l'application utilisatrice possèdent chacun leur propre table **TableEtats** dont le contenu peut différer.

Certaines erreurs occasionnelles se produisant lors des accès à un moteur KVS peuvent constituer du bruit si le problème est transitoire et ne correspond pas à un incident sérieux (perte de paquet réseau, microcoupure, ralentissement passager du système d'exploitation, etc.).

La sensibilité de l'état hystérétique à ce bruit est corrélée négativement à la valeur de **E**. Inversement, le délai de réaction de l'état hystérétique pour statuer un nœud est corrélé positivement à la valeur de **E**. Aux extrêmes, une valeur minimale (**E**=1) force l'état hystérétique égal à la condition technique, tandis qu'une valeur infinie (**E**=∞) empêche de statuer l'état hystérétique.

D'autre part, pendant l'indisponibilité d'un moteur KVS sur un nœud, une paire peut disparaitre dudit nœud ou manquer une création. Dans un tel cas et une fois le nœud redevenu disponible, une action de lecture de ladite paire sur ledit nœud ne produit pas d'erreur mais génère à tort une réponse négative (clé inexistante), appelée faux négatif.

Pour se prémunir contre de tels faux négatifs, l'invention considère une période dite de rémanence superposée à l'état hystérétique. Cette période de rémanence d'un nœud débute à l'instant où l'état hystérétique dudit nœud évolue de « indisponible » vers « disponible » et se termine après une durée **D**, constante configurable positive ou nulle [Tableau 4].

En d'autres termes, un nœud (n) est en période de rémanence si son état hystérétique est « disponible » et si (d_n < D) où (d_n) est le délai écoulé depuis la dernière inversion d'état du nœud (n) [560]. Ledit délai (d_n) se calcule comme la différence entre l'heure courante (m) et l'horodatage de la dernière inversion d'état du nœud (n) [552].

Toute réponse négative résultant de la lecture d'une paire sur un moteur KVS en rémanence, possède une probabilité non acceptable d'être un faux négatif, et à ce titre est ignorée [560 vers 556].

Ce procédé associant état hystérétique et période de rémanence est avantageux : il contribue à la cohérence des données et fonctionne en cercle vertueux. En effet, plus la sollicitation du moteur est intensive, plus l'état hystérétique est décidé rapidement. Tel ne serait pas le cas avec un procédé de supervision cyclique.

Les constantes configurables **E** et **D** influencent la performance et la fiabilité du procédé.

État hystérétique	**Période de rémanence**	**Description**
Indisponible	Non	L'état du service KVS est statué indisponible.
Disponible	Oui	L'état du service KVS est statué disponible. La durée écoulée depuis la dernière indisponibilité est inférieure à **D**. L'existence de faux négatifs est considérée comme probable.
	Non	L'état du service KVS est statué disponible. La durée écoulée depuis la dernière indisponibilité est supérieure ou égale à **D**. L'existence de faux négatifs est considérée comme non probable.

Tableau 4 – État hystérétique et période de rémanence d'un nœud/service KVS sur un nœud donné.

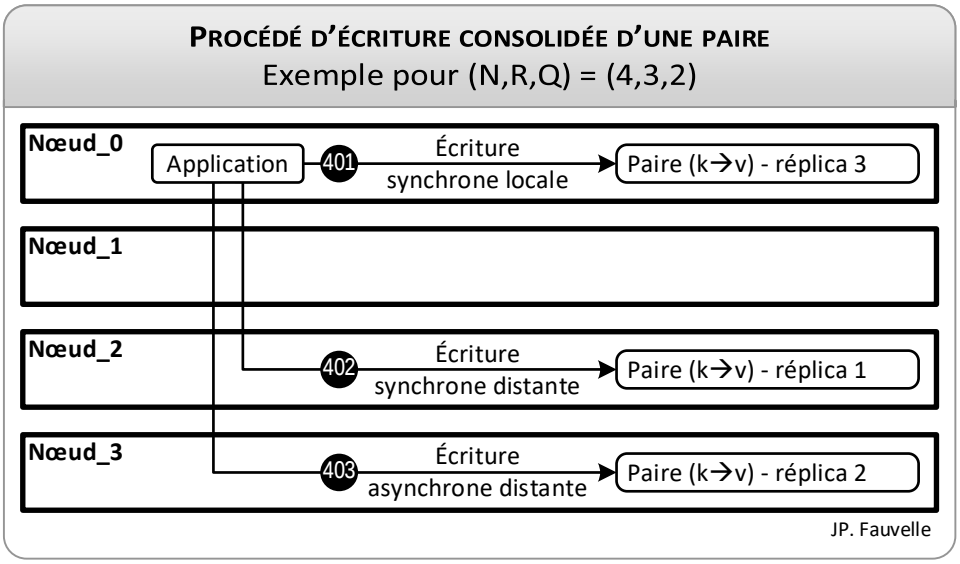

Figure 4 – Écriture consolidée d'une paire dans le RAM-Cloud.

1.2.6 PROCÉDÉ INTÉGRÉ PROBABILISTE RÉTROACTIF COMPLET DE LECTURE CONSOLIDÉE

1.2.6.1 LOI STATISTIQUE D'AMORTISSEMENT DES TRANSITIONS INTER-ÉTATS

La loi statistique d'amortissement fait partie de l'algorithme de lecture consolidée d'une paire dans le RAM-Cloud [§1.2.6.3] [Figure 5].

Lorsque l'état hystérétique d'un service/nœud KVS bascule de « disponible » vers « indisponible », l'algorithme de lecture cesse progressivement d'envoyer les requêtes vers ledit nœud afin de le décharger des sollicitations en lecture de façon linéairement décroissante sur une période d'amortissement **A**, paramètre configurable exprimant une durée positive ou nulle. Cette baisse progressive est avantageuse pour les raisons suivantes :

- Peu de temps après que l'état hystérétique d'un nœud soit « indisponible », la proportion des requêtes continuant à être transmises audit nœud est toujours importante, ce qui garantit une bonne réactivité du cycle d'hystérésis donc une transition rapide vers l'état « disponible » lorsque le nœud ne sera plus incidenté.
- À l'inverse, si l'état est « indisponible » depuis longtemps, alors la proportion des requêtes continuant à être transmises audit nœud est plus faible. Dans ce cas, le délai de réaction de la transition vers l'état « disponible » est plus important, sans que cela constitue un handicap puisque ledit délai reste proportionnellement négligeable au regard de la durée de l'indisponibilité.

Passé ce délai **A**, la sollicitation est maintenue à un plancher fixé arbitrairement à 1%. Ce flux non nul de requêtes est avantageux car il permet d'alimenter [511] le fonctionnement du compteur hystérétique en toute circonstance, ce qui dispense de mettre en place un système de supervision.

Symétriquement, lorsque l'état hystérétique d'un nœud KVS bascule de « indisponible » vers « disponible », l'algorithme de lecture recommence progressivement à envoyer les requêtes vers ledit moteur de façon linéairement croissante sur une période **A**. Cette hausse progressive est avantageuse, car elle permet de réduire le risque d'incident inhérent à une remise en charge brutale.

Passé ce délai **A** de remontée en charge, la sollicitation revient à son niveau de 100%.

Ce procédé d'amortissement requiert une fonction de probabilité multiparamétrique nommée **LoiTransition**, laquelle détermine [530] la probabilité (p) de lire un réplica sur un nœud donné, en fonction de l'état (e) hystérétique dudit nœud, du délai (d) écoulé depuis la dernière inversion d'état hystérétique dudit nœud, et de la constante d'amortissement A :

- Le délai (d) se calcule [552] comme le moment présent (m) moins l'horodatage (i) de la dernière inversion d'état hystérétique dudit nœud.
- L'état (e) hystérétique et l'horodatage (i) de la dernière inversion d'état sont calculés [510] par la fonction de comptage hystérétique puis mémorisé [512] dans une table **TableEtats** [520].

État (e) actuel du moteur KVS	Délai (d) depuis la dernière inversion d'état	Probabilité (p) que l'algorithme de lecture lise le réplica sur le moteur KVS
Indisponible	$d \in [0, A]$	$p = 1 - (1 - 0.01) * \frac{d}{A} = 1 - \frac{0.99 * d}{A}$
	$d \in]A, +\infty[$	$p = 0.01$ (1%)
Disponible	$d \in [0, A]$	$p = 0.01 + (1 - 0.01) * \frac{d}{A} = 0.01 + \frac{0.99 * d}{A}$
	$d \in]A, +\infty[$	$p = 1$ (100%)

Tableau 5 – Fonction LoiTransition d'amortissement des transitions d'états des moteurs KVS.

Remarque : la loi statistique d'amortissement ne concerne que les requêtes en lecture. Les requêtes en écriture ne sont pas régulées par une loi statistique, car elles sont négligeables comparativement aux lectures – par hypothèse d'usage d'un moteur NoSQL.

1.2.6.2 PRINCIPE RÉTROACTIF

Le procédé de lecture consolidée est rétroactif, comme décrit ci-après.

L'état hystérétique [510] est alimenté [511] occasionnellement par les écritures de réplicas, et majoritairement par les lectures de réplicas effectuées [555] par le procédé de lecture consolidée. En effet, la prédominance des lectures correspond au cas d'usage d'un moteur NoSQL.

L'état hystérétique alimente [512] la table **TableEtats** [520] qui à son tour alimente [521] le procédé de lecture consolidée. Ce dernier utilise l'information provenant de **TableEtats** d'une part pour déterminer la rémanence et d'autre part comme paramètre à la fonction de probabilité **LoiTransition**, les deux ayant pour effet de modifier le choix et le nombre de réplicas lus par le procédé de lecture consolidée, ce qui a pour effet d'impacter rétroactivement l'état hystérétique.

1.2.6.3 ALGORITHME DE LECTURE

1.2.6.3.1 PARAMÈTRES

L'application utilisatrice effectue une requête en lecture consolidée via l'invention [Figure 5, 500].

1. Lorsque l'application invoque la fonction de lecture, elle lui transmet [502] les paramètres suivants :
   - Obligatoirement, la clé (k) de la paire dont la lecture consolidée est demandée.
   - En option, un paramètre nommé « joker » (j) correspondant à l'âge en dessous duquel une donnée peut être considérée comme étant suffisamment fiable pour être admise en réponse définitive anticipée c.-à-d. sans chercher à lire d'autre réplicas ni à atteindre un quorum.

     Un joker de valeur nulle est privé d'effet car il ne peut exister une donnée d'âge négatif.

     Par défaut, ce paramètre est égal à zéro (procédé « joker » sans effet).
   - En option, un paramètre nommé « full_scan » (f) de type booléen. Lorsque ce paramètre est égal à VRAI, alors la lecture est effectuée sur la totalité des **R** réplicas c.-à-d. sans possibilité de terminer prématurément la lecture par atteinte d'un quorum ou déclenchement d'un joker.

     Par défaut, ce paramètre est FAUX (procédé « full_scan » sans effet).
2. Puis l'algorithme de lecture consolidée se poursuit par une étape d'initialisation [§1.2.6.3.2].

1.2.6.3.2 ÉTAPE D'INITIALISATION

1. L'algorithme de lecture consolidée procède aux initialisations suivantes [541] :
   - Une variable (r+) comptant les réponses positives est initialisée à zéro.
   - Une variable (r-) comptant les réponses négatives est initialisée à zéro.
   - Une variable (x') contenant la réponse de référence est initialisée à néant.
   - Une variable (h') contenant l'horodatage de référence est initialisée à zéro.
   - Une variable (q) contenant la valeur du quorum applicable pendant la lecture demandée, est initialisée à **Q**.
2. Puis, si le paramètre « full_scan » (f) est VRAI, alors [542] :
   - La variable (j) est initialisée à zéro, ce qui a pour effet de désactiver le procédé du « joker ».

- La variable (q) est initialisée à l'infini, ce qui a pour effet de désactiver le procédé du quorum positif et négatif.

3. Puis, une variable (ϕ) est initialisée avec la liste des R nœuds contenant les R réplicas associés à la clé (k), identifiés de façon déterministe par hachage sur la clé [§1.2.3].

   Cette file (ϕ) est ordonnée de manière aléatoire. Par exception, si l'application et les moteurs KVS partagent les mêmes nœuds et si le nœud depuis lequel l'application est en train d'invoquer la lecture [501] est présent dans la file, alors ce nœud local est positionné en tête de la file [543].

   Ce procédé est avantageux car il privilégie une première lecture locale. Or, dans le cas où l'application utilise le paramètre joker, l'algorithme de lecture peut aboutir à un résultat consolidé en lisant un seul réplica. L'association de cette lecture locale et du paramètre joker permet donc à l'algorithme de lecture consolidée d'aboutir à un résultat en effectuant une lecture unique et locale.

4. Enfin, l'algorithme de lecture consolidée se poursuit par une étape de lecture des réplicas [§1.2.6.3.3].

1.2.6.3.3 ÉTAPE DE LECTURES DES RÉPLICAS

L'algorithme de lecture consolidée, itérativement pour chaque nœud noté (n) prélevé [551] en tête de la file (ϕ) jusqu'à épuisement [550] de ladite file, exécute le procédé ci-après :

A. Appliquer **LoiTransition** [§1.2.6.1] :
   a. Lire [552] l'état (e_n) hystérétique [520] du nœud (n) provenant [521] de la table **TableEtats** opérée par le nœud nœud_0.
   b. Lire [552] l'horodatage (i_n) de la dernière inversion d'état [520] du nœud (n) provenant [521] de la table **TableEtats** opérée par le nœud nœud_0.
   c. Déterminer [552] le délai (d_n) écoulé depuis la dernière inversion d'état du nœud (n), calculé comme le moment présent (m) moins (i_n).
   d. Déterminer [553] la probabilité (p_n) de lire le réplica sur le nœud (n) calculée [531] comme LoiTransition(e_n, d_n, A).
   e. Effectuer [553] un tirage aléatoire (z) d'un nombre réel sur l'intervalle [0, 1].
   f. Si ($z > p_n$) [554] alors ignorer le nœud (n) et itérer [556] le procédé sur le nœud suivant dans (ϕ).
   g. Sinon ($z \leq p$) alors dérouler [555] le procédé ci-après avec le nœud (n).

B. Lire [555] le réplica noté `{ clé k ; donnée` v_n `= (objet` x_n`, horodatage` h_n`) }` associé à la clé (k) sur le nœud (n) :
   a. Si la tentative de lecture [555] du réplica sur le nœud (n) n'est pas terminée après une durée **T** alors interrompre la tentative de lecture, ignorer le nœud (n) et itérer [556] sur le prochain nœud dans la file (ϕ). **T** est une constante configurable. Ce procédé garantit la durée de lecture d'un réplica, ce qui à son tour garantit la durée du procédé de lecture consolidée, ce qui permet de traiter les requêtes en temps réel.
   b. Si la lecture [555] du réplica sur le nœud (n) produit une erreur alors ignorer le nœud (n) et itérer [556] sur le prochain nœud dans la file (ϕ).
   c. Si la lecture [555] du réplica sur le nœud (n) est négative [558] (clé inexistante) alors :
      1. Si l'état (e_n) hystérétique du nœud est « disponible » et si ($d_n < \mathbf{D}$) alors le nœud (n) est en **rémanence** [560] : ce présumé **faux négatif** ne contribue pas au résultat de l'algorithme, qui itère [556] sur le prochain nœud dans la file (ϕ).

2. Sinon, la variable (r-) comptant les réponses négatives est incrémentée [561].
3. Si (r- ≥ q) [562] alors le quorum des réponses négatives est atteint : l'algorithme termine prématurément la requête en statuant [563] une réponse négative.
4. Sinon (r- < q) [562] alors le quorum des réponses négatives n'est pas atteint : l'algorithme itère [556] sur le prochain nœud dans la file (ϕ).

d. Si la lecture [555] du réplica sur le nœud (n) est positive [557] alors :
   1. Si (j = 0) [570] alors le procédé du joker est inopérant : le traitement se poursuit au point 3 ci-après.
   2. Sinon (j > 0) [570] alors le procédé du joker est opérant et se déroule comme suit :
      i. Déterminer [571] la variable (a_n) correspondant à l'âge de la donnée lue sur le nœud (n), calculée comme le moment présent (m) moins l'horodatage (h_n) de la donnée.
      ii. Si (a_n < j) [572] alors la donnée lue sur le nœud (n) est plus jeune que la valeur du joker, elle est donc considérée comme fiable : l'algorithme met en œuvre le procédé du joker ce qui termine prématurément la requête en statuant [585] une réponse positive égale à la donnée lue sur le nœud (n).
      iii. Sinon (a_n ≥ j) [572] alors le traitement se poursuit au point 3 ci-après.
   3. Si l'horodatage (h_n) de la donnée lue sur le nœud (n) est inférieur [580] à l'horodatage (h') de référence, alors la donnée lue est obsolète car plus ancienne que la donnée de référence, ce qui constitue un cas d'incohérence positive des réplicas : l'algorithme ignore la lecture du nœud (n) et itère [556] sur le prochain nœud dans la file (ϕ).
   4. Si l'horodatage (h_n) de la donnée lue sur le nœud (n) est supérieur [581] à l'horodatage (h') de référence, alors le compteur (r+) des réponses positives porte sur une donnée obsolète, ce qui constitue un cas d'incohérence positive des réplicas : l'algorithme remet à zéro [584] le compteur (r+).
   5. Incrémenter [582] le compteur (r+) des réponses positives.
   6. Copier [582] l'objet (x_n) dans l'objet de référence (x').
   7. Copier [582] l'horodatage (h_n) dans l'horodatage de référence (h').
   8. Si (r+ ≥ q) [583] alors le quorum des réponses positives est atteint : l'algorithme termine prématurément la requête en statuant [585] une réponse positive égale à la donnée de référence x'.
   9. Sinon (r+ < q) [583] alors le quorum des réponses positives n'est pas atteint : l'algorithme itère [556] sur le prochain nœud dans la file (ϕ).

Une fois la file (ϕ) épuisé [550] et si l'algorithme n'a pas pu statuer une réponse, alors [590] :

1. Si (r+ > 0) [590] alors il existe au moins une réponse positive : l'algorithme statue [585] une réponse positive égale à la donnée de référence x'.
2. Si (r- > 0) [591] alors il existe au moins une réponse négative : l'algorithme statue [563] une réponse négative.
3. Sinon, l'algorithme statue [592] une réponse en erreur.

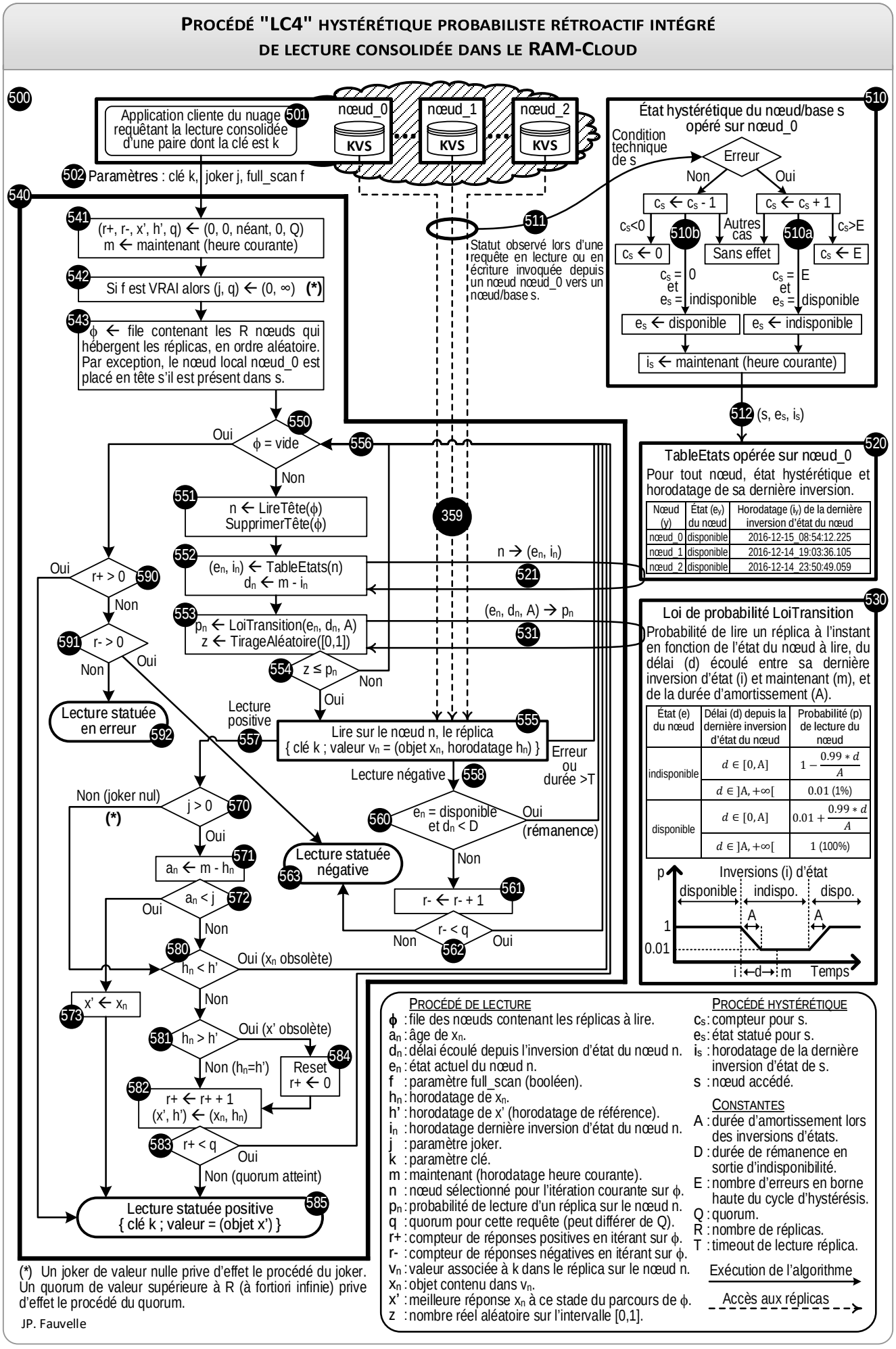

Figure 5 – Procédé « LC4 » hystérétique probabiliste rétroactif intégré de lecture consolidée d'une paire.

1.2.6.4 EXEMPLES DE CINÉMATIQUES DE LECTURE

Les exemples ci-après s'appuient sur les hypothèses suivantes :

- Conformément au cas nominal, les nœuds clients et les nœuds de données sont confondus.
- **N** = 4 ; **R** = 3 ; **Q** = 2.
- Par souci de clarté, les réplicas sont accédés dans un ordre déterministe c.-à-d. sans respecter le caractère aléatoire ni la loi statistique de l'amortissement [§1.2.6.1].

1.2.6.4.1 EXEMPLE LFP1 - LECTURE FULL_SCAN POSITIVE UNANIME, CAS NOMINAL

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica.	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica	1→2	0	Nœud_3	Disponible	Positif	xxx	t1
4	Lire la paire sur le troisième réplica.	2→3	0	Nœud_1	Disponible	Positif	xxx	t1
5	Le résultat est la donnée possédant le plus grand horodatage. La requête en lecture est statuée positive à l'unanimité (r+ = 3) et le résultat est : `{ clé k ; donnée xxx }.`	3	0					

Tableau 6 – Cinématique de lecture / Exemple LFP1.

1.2.6.4.2 EXEMPLE LFP2 – LECTURE FULL_SCAN POSITIVE MAJORITAIRE, OBSOLESCENCE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica.	0→1	0	Nœud_0	Disponible	Positif	yyy	t2
3	Lire la paire sur le second réplica.	1→2	0	Nœud_3	Disponible	Positif	yyy	t2
4	Lire la paire sur le troisième réplica. t1 < t2 ➔ donnée xxx obsolète donc ignorée.	2	0	Nœud_1	Disponible	Positif	xxx	t1 (t1< t2)
5	Le résultat est la donnée possédant le plus grand horodatage. La requête en lecture est statuée positive à la majorité (r+ = 2) et le résultat est : `{ clé k ; donnée yyy }.`	2	0					

Tableau 7 – Cinématique de lecture / Exemple LFP2.

1.2.6.4.3 EXEMPLE LFP3 - LECTURE FULL_SCAN POSITIVE MINORITAIRE, OBSOLESCENCE ET FAUTE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				Lecture de la donnée				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica.	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica. t1 < t2 ➔ Donnée xxx obsolète donc ignorée.	1→0 0→1	0	Nœud_3	Disponible	Positif	yyy	t2 (t1< t2)
4	Lire la paire sur le troisième réplica.	1	0	Nœud_1	Indisponible	Erreur		
5	Le résultat est la donnée possédant le plus grand horodatage. La requête en lecture est statuée positive à la minorité (r+ = 1) et le résultat est : `{ clé k ; donnée yyy }.`	1	0					

Tableau 8 – Cinématique de lecture / Exemple LFP3.

1.2.6.4.4 EXEMPLE LFP4 - LECTURE FULL_SCAN POSITIVE MINORITAIRE, FAUTES MULTIPLES

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				Lecture de la donnée				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica.	0	0→1	Nœud_0	Disponible	Négatif		
3	Lire la paire sur le second réplica.	0→1	1	Nœud_3	Disponible	Positif	yyy	t2
4	Lire la paire sur le troisième réplica.	1	1	Nœud_1	Indisponible	Erreur		
5	Le résultat est la donnée possédant le plus grand horodatage. La requête en lecture est statuée positive par minorité (r+ = 1) et le résultat est : `{ clé k ; donnée yyy }.`	1	1					

Tableau 9 – Cinématique de lecture / Exemple LFP4.

1.2.6.4.5 EXEMPLE LFN1 - LECTURE FULL_SCAN NÉGATIVE UNANIME, CAS NOMINAL

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica.	0	0→1	Nœud_0	Disponible	Négatif		
3	Lire la paire sur le second réplica.	0	1→2	Nœud_3	Disponible	Négatif		
4	Lire la paire sur le troisième réplica.	0	2→3	Nœud_1	Disponible	Négatif		
5	La requête en lecture est statuée négative à l'unanimité (r- = 3).	0	3					

Tableau 10 – Cinématique de lecture / Exemple LFN1.

1.2.6.4.6 EXEMPLE LFN2 - LECTURE FULL_SCAN NÉGATIVE MINORITAIRE, FAUTES MULTIPLES

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				Lecture de la donnée				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica.	0	0→1	Nœud_0	Disponible	Négatif		
3	Lire la paire sur le second réplica.	0	1	Nœud_3	Indisponible	Erreur		
4	Lire la paire sur le troisième réplica.	0	1	Nœud_1	Indisponible	Erreur		
5	La requête en lecture est statuée négative à la minorité (r- = 1).	0	1					

Tableau 11 – Cinématique de lecture / Exemple LFN2.

1.2.6.4.7 EXEMPLE LJP1 - LECTURE JOKER POSITIVE RAPIDE, CAS NOMINAL

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. age_v ← HeureCourante() – t1 age_v < joker ➔ Joker applicable donc donnée xxx acceptée ➔ Fin immédiate de la requête en lecture.	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	La requête en lecture est statuée positive immédiatement à la minorité (r+ = 1) et le résultat est : `{ clé k ; donnée xxx }`	1	0					

Tableau 12 – Cinématique de lecture / Exemple LJP1.

1.2.6.4.8 EXEMPLE LJP2 - LECTURE JOKER POSITIVE, REPLI VERS QUORUM

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. age_v ← HeureCourante() – t1 age_v ≥ joker ➔ Joker non applicable donc repli vers le procédé du quorum : r+ ← 1 pour la donnée xxx.	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica. age_v ← HeureCourante() – t1 age_v ≥ joker ➔ Joker non applicable donc repli vers le procédé du quorum : r+ ← 2 = **Q** ➔ Atteinte du quorum donc fin immédiate de la requête en lecture.	1→2	0	Nœud_3	Disponible	Positif	xxx	t1
4	La requête en lecture est statuée positive à la majorité (r+ = 2) et le résultat est : `{ clé k ; donnée xxx }`	2	0					

Tableau 13 – Cinématique de lecture / Exemple LJP2.

1.2.6.4.9 EXEMPLE LJP3 - LECTURE JOKER POSITIVE, REPLI VERS QUORUM ET OBSOLESCENCE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. age_v ← HeureCourante() – t1 age_v ≥ joker ➔ Joker non applicable donc repli vers le procédé du quorum : r+ ← 1 pour la donnée xxx.	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica. age_v ← HeureCourante() – t2 age_v ≥ joker ➔ Joker non applicable donc repli vers le procédé du quorum : t1 < t2 ➔ Redémarrage du compteur r+ pour la donnée yyy. r+ ← 1 pour la donnée yyy.	1→0 0→1	0	Nœud_3	Disponible	Positif	yyy	t2 (t1< t2)
4	Lire la paire sur le troisième réplica. age_v ← HeureCourante() – t1 age_v ≥ joker ➔ Joker non applicable donc repli vers le procédé du quorum : t1 < t2 ➔ Donnée obsolète donc non prise en compte dans le quorum.	1	0	Nœud_1	Disponible	Positif	xxx	t1
5	La requête en lecture est statuée positive à la minorité (r+ = 1) et le résultat est : `{ clé k ; donnée yyy }`	1	0					

Tableau 14 – Cinématique de lecture / Exemple LJP3.

1.2.6.4.10 EXEMPLE LQP1 - LECTURE QUORUM POSITIF MAJORITAIRE RAPIDE, CAS NOMINAL

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				Lecture de la donnée				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. r+ ← 1 pour la donnée xxx.	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica. r+ ← 2 = **Q** ➔ Atteinte du quorum donc fin immédiate de la requête en lecture.	1→2	0	Nœud_3	Disponible	Positif	xxx	t1
4	La requête en lecture est statuée positive à la majorité (r+ = 2) et le résultat est : `{ clé k ; donnée xxx }`	2	0					

Tableau 15 – Cinématique de lecture / Exemple LQP1.

1.2.6.4.11 EXEMPLE LQP2 - LECTURE QUORUM POSITIF MAJORITAIRE, OBSOLESCENCE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. r+ ← 1 pour la donnée xxx	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica. t1 < t2 ➔ Redémarrage du compteur r+ pour la donnée yyy. r+ ← 1 pour la donnée yyy.	1→0 0→1	0	Nœud_3	Disponible	Positif	yyy	t2 (t1< t2)
4	Lire la paire sur le troisième réplica. r+ ← 2 = **Q** ➔ Atteinte du quorum donc fin immédiate de la requête en lecture.	1→2	0	Nœud_1	Disponible	Positif	yyy	t2
5	La requête en lecture est statuée positive à la majorité (r+ = 2) et le résultat est : `{ clé k ; donnée yyy }`	2	0					

Tableau 16 – Cinématique de lecture / Exemple LQP2.

1.2.6.4.12 EXEMPLE LQP3 - LECTURE QUORUM POSITIF MINORITAIRE, OBSOLESCENCE ET FAUTE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. r+ ← 1 pour la donnée xxx	0→1	0	Nœud_0	Disponible	Positif	xxx	t1
3	Lire la paire sur le second réplica. t1 < t2 ➔ Redémarrage du compteur r+ pour la donnée yyy. r+ ← 1 pour la donnée yyy.	1→0 0→1	0	Nœud_3	Disponible	Positif	yyy	t2 (t1< t2)
4	Lire la paire sur le troisième réplica. r- ← 1	1	1	Nœud_1	Disponible	Négatif		
5	La requête en lecture est statuée positive à la minorité (r+ = 1) et le résultat est : `{ clé k ; donnée yyy }`	1	1					

Tableau 17 – Cinématique de lecture / Exemple LQP3.

1.2.6.4.13 EXEMPLE LQN1 - LECTURE QUORUM NÉGATIF MAJORITAIRE RAPIDE, CAS NOMINAL

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				**Lecture de la donnée**				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. r- ← 1.	0	0→1	Nœud_0	Disponible	Négatif		
3	Lire la paire sur le second réplica. r- ← 2 = **Q** ➔ Atteinte du quorum donc fin immédiate de la requête en lecture.	0	1→2	Nœud_3	Disponible	Négatif		
4	La requête en lecture est statuée négative à la majorité (r- = 2).	0	2					

Tableau 18 – Cinématique de lecture / Exemple LQN1.

1.2.6.4.14 EXEMPLE LQN2 - LECTURE QUORUM NÉGATIF MAJORITAIRE, RÉMANENCE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				Lecture de la donnée				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. r- ← 1.	0	0→1	Nœud_0	Disponible	Négatif		
3	Lire la paire sur le second réplica. La lecture négative ne contribue pas au quorum négatif car nœud_3 est en rémanence (lectures négatives présumées non fiables).	0	1	Nœud_3	**Rémanence**	Négatif		
4	Lire la paire sur le troisième réplica. r- ← 2 = **Q** ➔ Atteinte du quorum donc fin immédiate de la requête en lecture.	0	1→2	Nœud_1	Disponible	Négatif		
5	La requête en lecture est statuée négative à la majorité (r- = 2).	0	2					

Tableau 19 – Cinématique de lecture / Exemple LQN2.

1.2.6.4.15 EXEMPLE LQN3 - LECTURE QUORUM NÉGATIF MAJORITAIRE, CAS MIXTE

Application s'exécutant sur le nœud client nœud_0, effectuant une requête en lecture vers une donnée dont la clé vaut k				Lecture de la donnée				
Étape	**Action**	**r+**	**r-**	**Nœud accédé**	**État nœud**	**Statut lecture**	**Donnée lue**	**Horo-datage**
1	Déterminer la file (ϕ) comprenant les **R** nœuds hébergeant les **R** réplicas associés à la clé (k). Si le nœud local nœud_0 d'où provient la requête émise par l'application est présent dans la file, alors déplacer nœud_0 en tête de file. Par souci de lisibilité, les réplicas sont accédés sans respecter le caractère aléatoire ni la loi statistique de l'amortissement de transition [§1.2.6.1]. ϕ ← {nœud_0, nœud_3, nœud_1}.	0	0					
2	Lire la paire sur le premier réplica. r- ← 1	0	0→1	Nœud_0	Disponible	Négatif		
3	Lire la paire sur le second réplica. r+ ← 1 pour la donnée xxx.	0→1	1	Nœud_3	Disponible	Positif	xxx	t1
4	Lire la paire sur le troisième réplica. r- ← 2 = **Q** ➔ Atteinte du quorum donc fin immédiate de la requête en lecture.	1	1→2	Nœud_1	Disponible	Négatif		
5	La requête en lecture est statuée négative à la majorité (r- = 2).	1	2					

Tableau 20 – Cinématique de lecture / Exemple LQN3.

1.2.6.5 PERFORMANCE

Ce procédé associant version, rémanence, joker, quorum positif, quorum négatif débrayable en période de rémanence, permet de gérer tous les cas d'incohérence positive/négative des données et offre simultanément une résilience, une cohérence, et des performances maximales.

Complexité moyenne en temps dans les cas nominaux	Temps de réponse vu par l'application				
	Lectures locales			Lectures distantes	
	Quantité	Probabilité		Quantité	Probabilité
Lecture positive avec joker.	1	$\frac{R}{N}$	+	1	$1-\frac{R}{N}$
Lecture positive sans joker.	1	$\frac{R}{N}$	+	$Q-1=\left\lceil\frac{R}{2}\right\rceil-1$	1
				+	
Lecture négative.				1	$1-\frac{R}{N}$

Tableau 21 – Complexité moyenne en temps des accès en lecture.

Avec des hypothèses standards, ce procédé permet d'atteindre la meilleure complexité en temps prévue par la théorie, sans pénaliser la résilience, quels que soient le volume de la base et le nombre de nœuds.

Non seulement une lecture globale positive, dans le cas nominal avec joker, se réalise via une unique lecture unitaire d'un réplica, mais de plus cette lecture est locale dans $\frac{R}{N}$ % des cas et se fait en mémoire.

Sur des serveurs standards, la lecture d'une paire s'effectue en un temps moyen d'environ **100 µs**, qui dépend de la qualité d'implémentation et du langage de programmation.

Ce temps peut être ramené à **10 µs**, voire moins, grâce à l'utilisation de technologies complémentaires.

Ces technologies sont, par exemple, les solutions HSE (*High Speed Ethernet*) ou InfiniBand, associées au protocole RDMA (*Remote Direct memory Access*) ou RoCE (*RDMA over Convergent Ethernet*) qui autorisent des accès directs en mémoire en court-circuitant le système d'exploitation.

1.2.7 RÉSILIENCE

L'invention est robuste à **r**-1 pannes simultanées et répare automatiquement les incohérences de données qui en découlent, ce qui permet de maintenir le niveau de résilience même lorsque des pannes se répètent.

Pour un couple (**R**, **N**) donné, un même nombre de pannes simultanées ne produit pas le même effet selon la localisation des pannes sur les nœuds car les **R** réplicas d'une paire se situent sur des nœuds consécutifs dans la liste des nœuds.

Le tableau ci-après présente la résilience moyenne des données pour des valeurs nominales de **R** et **N**, et pour un nombre **P** de pannes simultanées conforme aux limites d'utilisation (**P** < **R**) mais également hors des limites [Figure 6].

	N=2		N=3		N=4		N=5		N=6	
	Pannes	**Résil. moy.**	**Pannes**	**Résil. moy.**	**Pannes**	**Résil. moy.**	**Pannes**	**Résil. moy.**	**Pannes**	**Résil. moy.**
R=2	1	100%	1	100%	1	100%	1	100%	1	100%
	2	0%	2	67%	2	83%	2	90%	2	93%
			Pannes	**Résil. moy.**	**Pannes**	**Résil. moy.**	**Pannes**	**Résil. moy.**	**Pannes**	**Résil. moy.**
R=3	N/A		1	100%	1	100%	1	100%	1	100%
			2	100%	2	100%	2	100%	2	100%
			3	0%	3	75**%**	3	90%	3	(*) 95%

Tableau 22 – Résilience moyenne en fonction du nombre de pannes et de nœuds.

$$(*)\ Exemple\ :\ calcul\ de\ la\ résillience\ moyenne = \frac{6 * \frac{N-1}{N} + 14 * \frac{N}{N}}{C_N^3} = \frac{20 - \frac{6}{N}}{\frac{N!}{3!(N-3)!}} = 0.95$$

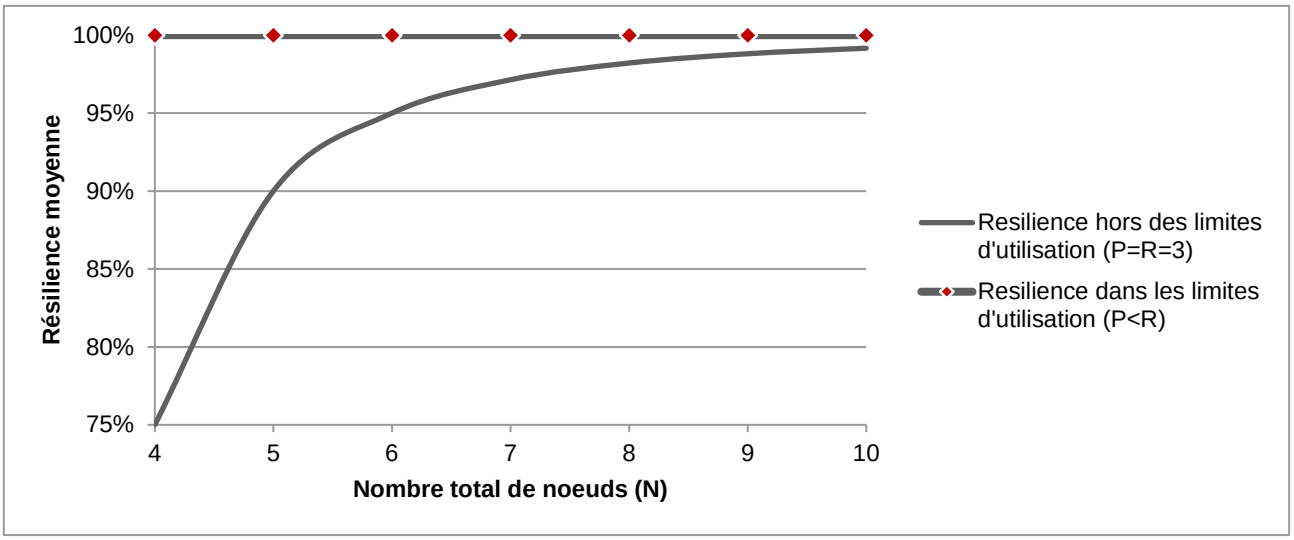

Figure 6 – Résilience moyenne dans les limites d'utilisation, et hors limites (triples pannes).

1.2.8 ARBITRAGE DYNAMIQUE AUTOMATIQUE SOUS LE CONTRÔLE DE L'APPLICATION

L'algorithme de lecture consolidée permet un **arbitrage dynamique automatique** entre la performance et la fiabilité.

En complément de cet arbitrage automatique :

- Le paramètre « full_scan » permet à l'application d'exercer sur l'invention un contrôle qui permet de maximiser la fiabilité avec une granularité fine par requête.
- Le paramètre « joker » permet à l'application d'exercer sur l'invention un contrôle qui permet de maximiser la performance avec une granularité fine par requête.

1.3 AMÉLIORATIONS POSSIBLES

Doter l'invention d'une persistance sur disque :

- D'un mécanisme qui écrit séquentiellement dans un fichier les modifications de données.
- D'un mécanisme lui permettant de régénérer les données dans les KVS en mémoire, à partir des fichiers précités.

2 PÉRIMÈTRE DE L'INVENTION

2.1 DOMAINE TECHNIQUE DE L'INVENTION

L'invention est un *framework* de **composant logiciel** embarqué dans une application, qui, associé à un service standard quelconque de type KVS, apporte à l'application les fonctions d'un moteur NoSQL en mémoire, temps réel, s'adaptant automatiquement au contexte de production pour apporter à l'application le plus haut niveau de résilience / cohérence / performance.

2.2 CARACTÉRISTIQUES ESSENTIELLES DE L'INVENTION

Caractéristiques de l'invention	Importance
Algorithme de lecture consolidée, s'appuyant sur l'état hystérétique, la période de rémanence, la loi statistique d'amortissement des transitions d'états.	**Essentielle**
Index multiplanaires.	**Essentielle**

Tableau 23 – Caractéristiques essentielles et secondaires de l'invention.

2.3 COMPARAISON DE L'INVENTION AVEC UN PRODUIT PHARE DU MARCHÉ

Une solution existante « CouchBase » semble en apparence proche. Le tableau ci-après liste quelques différences essentielles sans que la liste soit exhaustive :

Fonctionnalité	L'invention	CouchBase
Risque de lecture inconsistante ?	Non	Oui
Risque de faux négatifs lors des lectures de réplicas ?	Non	Oui
Délai de *failover* en cas d'incident ?	L'application contrôle le comportement de l'invention : • Soit le *failover* est instantané (E=1). • Soit le *failover* intervient dans un délai d'autant plus court que la sollicitation est importante (cercle vertueux), avec vitesse paramétrable (E>1). Exemple, pour E=5 et pour un débit de 100 requêtes par seconde, le *failover* intervient en 5 millisecondes.	Détection en 30 s
Retour au nominal après un incident ?	Le flux revient progressivement afin de réduire le risque de nouvel incident en cas d'instabilité.	Retour brutal
Besoin d'administrer un moteur NoSQL ?	Non	Oui
Persistance disque ?	Non (évolution possible).	Oui
Compatible tous KVS ?	Oui	Non
Fonctionnement temps réel ?	Oui - Complexité temporelle en O(1) et délais garantis.	Non

Tableau 24 – État de la technique, éléments de différentiation.

2.4 APPLICATIONS PRIVILÉGIÉES

Les applications de type Internet et/ou *proxy* et/ou *reverse-proxy* :

- Traitant un grand nombre de requêtes (exemple : milliards de requêtes par minute).
- Nécessitant une capacité de mise à l'échelle horizontale et à chaud.
- Exigeant un accès aux données en temps réel ou avec une très faible latence (de quelques microsecondes à quelques millisecondes).
- Requérant un haut niveau de résilience / performance / cohérence, ajustable à la volée selon les données accédées.

3 DESCRIPTION FORMELLE DE L'INVENTION

3.1 DOMAINE TECHNIQUE DE L'INVENTION

Le domaine de l'invention est celui des réseaux et systèmes informatiques.

Plus précisément, l'invention concerne un procédé et un système d'échange et de stockage de données informatiques en temps réel entre une pluralité de clients informatiques ou flotte, et une pluralité de serveurs informatiques ou parc.

L'invention permet notamment de router une importante quantité de flux de toute flotte d'équipements informatiques fixes ou mobiles.

L'invention permet notamment de stocker une importante quantité d'information, et d'y accéder en temps réel avec une fiabilité et une performance maximales et ajustables, particulièrement lors des accès en lecture, lesquels constituent le principal cas d'usage.

L'invention concerne notamment le domaine de la télématique, le domaine bancaire et monétique, le domaine de la téléphonie ou tout autre domaine comprenant des échanges de données informatiques entre une flotte de clients informatiques et un parc de serveurs informatiques : distribution de tickets, contrôle d'accès (badges, tickets, biométrie), bornes de jeux, box internet, site internet de commerce en ligne, collecte d'information, alarmes, compteurs d'eau ou d'énergie, badgeuses, capteurs, objets connectés.

3.2 ETAT DE LA TECHNIQUE

L'échange de données informatiques en temps réel entre d'une part une flotte comportant un grand nombre de clients informatiques et d'autre part un parc de serveurs ou nœuds de traitement des données informatiques, est complexe à réaliser. En effet, l'important volume de données à traiter chaque seconde nécessite un routage des données informatiques dans une durée brève et garantie afin de ne pas pénaliser l'échange.

La qualité d'un système d'échange de données informatiques performant se traduit notamment par :

- Un haut niveau de performance sur les traitements d'échanges ;
- Un haut niveau de fiabilité sur les traitements d'échanges, c'est-à-dire la capacité à fonctionner en cas d'incident ;
- Un haut niveau de performance sur les accès aux données ;
- Un haut niveau de fiabilité sur les accès aux données, c'est-à-dire la capacité pour un système à accéder aux données et à les conserver, sans perte ni altération.

Il est connu de l'état de l'art des techniques d'échange de données informatiques se basant sur un routeur stockant les informations de routage dans une base de données de type NoSQL (« *Not Only Structured Query Language* » en anglais), et/ou de type IMDBMS (« *In-Memory DataBase Management System* » en anglais) c'est-à-dire où le stockage s'effectue dans la mémoire vive de serveurs.

Le principal inconvénient de ces techniques est qu'elles ne permettent pas d'obtenir un système d'échange de données informatiques très performant et très fiable pour l'échange d'une multitude de données, du moins pas simultanément sur les deux qualités ni dans tous les cas d'usage. En effet, la compétition entre les qualités d'un système d'échange de données informatiques, à savoir la performance et la fiabilité, est généralement arbitrée de façon figée par

conception ou au mieux par paramétrage, en privilégiant une qualité au détriment de l'autre ou certains cas d'usage au détriment d'autres.

D'autre part, ces techniques nécessitent l'installation d'un progiciel de base de données et de son écosystème, de le superviser, de le sauvegarder, de l'administrer, et plus généralement de l'exploiter, ce qui occasionne un supplément de coût et charge de travail.

Afin d'obtenir un système d'échange de données résilient, c'est-à-dire ayant la capacité de continuer à fonctionner normalement, sans aucune interruption, malgré l'apparition d'une défaillance ou de plusieurs sur les serveurs ou nœuds du système, il est connu de l'état de l'art de stocker plusieurs exemplaires d'une même donnée. Ces exemplaires identiques, couramment appelés réplicas, sont généralement stockés dans différentes bases de données hébergées sur des serveurs distincts.

Un inconvénient de cette technique est qu'un incident sur un serveur stockant un des réplicas peut provoquer des erreurs dans le traitement des données. En effet, la base de données du serveur incidentée n'étant plus à jour, une donnée obsolète peut être lue. Il est également possible qu'une donnée manque, si un incident révolu a entrainé la disparation du réplica ou empêché sa création. Cette information absente à tort, ou faux négatif, peut faire croire que l'information n'existe pas alors qu'elle est manquante en raison d'un incident révolu dont la conséquence persiste.

Un autre inconvénient de cette technique est que la remise en service brutale d'un serveur après une indisponibilité peut provoquer de nouveaux incidents.

Un autre inconvénient de cette technique est le délai de remise en service automatique d'un serveur après une indisponibilité. En effet, le diagnostic de l'état d'un serveur dépend généralement d'un processus de supervision externe cyclique donc sans détection entre deux cycles.

3.3 OBJECTIFS DE L'INVENTION

La présente invention vise à remédier à tout ou partie des inconvénients de l'état de la technique cités précédemment.

Un des principaux objectifs de l'invention est de proposer un procédé d'échange et de stockage de données informatiques qui permet un arbitrage dynamique automatique entre les qualités de performance et de fiabilité.

Un autre objectif principal de l'invention est de proposer, lors des accès en lecture aux données, un procédé qui permet un arbitrage dynamique automatique entre la performance et la fiabilité, et qui permet d'influencer instantanément ledit arbitrage dans un sens ou dans l'autre avec une granularité par requête afin de tenir compte des particularités de chaque donnée et de son contexte d'utilisation.

Un autre objectif principal de l'invention est de proposer un procédé qui permet d'opérer un transfert de charge progressif entre les nœuds de la base de données, depuis des nœuds lorsque ils sont statués indisponibles puis vers ces mêmes nœuds lorsque ils redeviennent disponibles.

Un autre objectif principal de l'invention est de proposer un procédé qui permet de tenir compte des incidents durables et d'ignorer ceux qui sont éphémères.

Un des objectifs de l'invention est également de proposer un procédé qui permet de traiter un très grand volume de données informatiques, en temps réel c'est-à-dire avec des temps de traitement simultanément faibles et garantis.

Un autre objectif de l'invention est de proposer un procédé possédant une capacité de mise à l'échelle (« *scalability* » en anglais) horizontale et à chaud de type « big data **»**.

Un autre objectif de l'invention est de proposer un procédé qui soit fortement résilient, c'est-à-dire qui puisse continuer à fonctionner normalement et sans aucune interruption lorsque des nœuds deviennent indisponibles suite à une panne ou à un arrêt volontaire pour maintenance par exemple.

Un autre objectif de l'invention est de proposer un procédé basé sur des informations de routage cohérentes et non obsolètes.

Un autre objectif de l'invention est d'obtenir un procédé qui puisse s'adapter à n'importe quel type d'architecture du système d'échange de données.

Un autre objectif de l'invention est d'obtenir un procédé ne nécessitant pas l'installation d'un progiciel, tel qu'un moteur NoSQL, ni de son environnement.

3.4 EXPOSÉ DE L'INVENTION

Ces objectifs, ainsi que d'autres qui apparaîtront par la suite sont atteints à l'aide d'un procédé d'échange de données informatiques en temps réel entre un logiciel opéré sur un client informatique et un logiciel opéré sur un serveur informatique dit serveur de traitement, le client informatique et le serveur de traitement étant compris dans un système d'échange de données comprenant une pluralité de clients informatiques ou flotte, une pluralité de serveurs de traitement ou parc, et des moyens de routage permettant de relier les clients aux serveurs de traitement, les moyens de routage étant connectés à des moyens de stockage de données, les moyens de routage comprenant au moins un serveur de routage.

Le procédé objet de l'invention permet d'échanger des données informatiques, entre une flotte ou pluralité de clients informatiques, et un parc ou pluralité de serveurs aptes à traiter les données informatiques. Les échanges de données informatiques peuvent s'effectuer indépendamment de manière sécurisée ou non sécurisée. Les données informatiques peuvent être associées à des requêtes informatiques. Les clients informatiques peuvent être par exemple des ordinateurs personnels, des téléphones mobiles ou des automates bancaires. Les automates bancaires sont par exemple des terminaux de paiements électroniques (acronyme TPE), des distributeurs automatiques de billets (DAB), des guichets automatiques de banque (GAB), ou des bornes en libre service. Les clients informatiques peuvent être également des bornes de jeux, des bornes de distribution de tickets (transports, parking, etc.), des box d'accès à Internet, des systèmes de collecte d'information (alarmes, compteurs d'eau ou d'énergie, badgeuses, biométrie), des objets connectés, des boitiers de routage, etc. Les serveurs de traitement peuvent quant à eux être hébergés dans des centres de données couramment appelés « *data-centers* ». Les données provenant d'un client informatique sont traitées par un serveur de traitement qui est compatible avec la version du logiciel à l'origine des données informatiques. Le logiciel du client informatique peut être par exemple un navigateur internet, ou un progiciel propriétaire.

Il convient de noter que le serveur de routage peut également être appelé nœud informatique de routage ou nœud de routage ou nœud.

Le procédé d'échange entre des clients informatiques et des serveurs informatiques comprend un procédé dit de nuage. Ce procédé de nuage permet d'opérer un nuage contenant des groupes de données, chaque groupe étant associé à une clé, et d'y accéder en lecture comme en écriture.

Il convient de noter que tout groupe de données noté objet x associé à une clé notée k peut être représenté par une paire notée { clé k ; valeur v = (objet x, …) }, la valeur v comprenant au moins l'objet x, la clé k étant soit une clé dite primaire composée d'une seule information soit une clé dite composite agrégeant plusieurs informations.

Selon l'invention, le procédé d'échange de données informatiques effectue une lecture et/ou une écriture d'au moins une information de routage stockée dans un groupe de données sauvegardé dans au moins une base de données des moyens de stockage de données, le groupe de données étant associé à une clé comprenant l'identification du client informatique, les moyens de stockage de données comprenant un système informatique en nuage comprenant une pluralité de nœuds informatiques, la base de données étant gérée par un moteur de base de données opérant sur un des nœuds informatiques.

Il convient de noter qu'une base de données peut également être appelée une base, et qu'un moteur de base de données peut également être appelé un moteur.

Il convient de noter que le procédé d'échange, au même titre que tout autre procédé accédant au nuage, est dit client du nuage puisqu'il invoque des requêtes d'accès aux données auxquelles le procédé du nuage répond.

Ainsi, le procédé d'échange de données informatiques accède rapidement aux informations de routage qui sont avantageusement stockées dans un système informatique en nuage, également connu sous le nom de « *cloud* ». Le système informatique en nuage comprend une pluralité de serveurs informatiques reliés par l'intermédiaire d'un réseau, partageant ainsi leurs ressources informatiques. Chaque serveur informatique correspond à un nœud informatique du réseau formant le système informatique en nuage. Les informations de routage associées à une identification d'un client informatique comprennent les routes par défaut ou « default routes » alias « DR » en anglais, les ordres de déroutage ou « next routes » alias « NR » en anglais, ainsi que les routes courantes ou « hot routes » alias « HR » en anglais.

Les routes par défaut pour un client informatique correspondent aux routes possibles entre ce client informatique et un serveur de traitement compatible avec la version du logiciel du client informatique.

Les ordres de déroutage indiquent pour un client informatique, quelle est la nouvelle route à suivre. Les ordres de déroutage peuvent également indiquer si les données informatiques provenant d'un client informatique en particulier sont refusées. Un caractère d'immédiateté de l'ordre de déroutage peut être inclus dans chaque ordre de déroutage. En d'autres termes, le caractère d'immédiateté indique une priorité ou un délai pour chaque ordre de déroutage.

Chaque route courante est associée à une identification d'un client informatique et indique le serveur de traitement qui est actuellement associé à l'identification dudit client. Les routes courantes peuvent également indiquer l'identification du client, la version du logiciel du client, et l'horodatage de la dernière requête reçue.

Il convient de noter qu'une version du logiciel correspond à un état de développement du logiciel. Une version du logiciel est généralement associée à une numérotation ou à un horodatage qui permet d'identifier la version, voire à un nom symbolique.

*****

Dans des modes de réalisation particuliers de l'invention, le procédé d'échange de données informatiques est mis en œuvre par au moins un processus d'une application informatique, le processus étant opéré sur le serveur de routage et/ou sur un nœud informatique du système informatique en nuage.

Ainsi, l'application informatique mettant en œuvre le procédé objet de l'invention comprenant plusieurs processus, également appelés instances, est dite multi-instanciée.

Il convient de noter que l'application de routage, au même titre que toute autre application accédant au nuage et que toute instance d'une telle application, est dite cliente du nuage puisqu'elle invoque des requêtes d'accès aux données auxquelles le nuage répond.

*****

Dans des modes de réalisation particuliers de l'invention, le système informatique en nuage est un RAM-Cloud ou un SSD-Cloud.

Le RAM-Cloud est un système informatique en nuage dont les données sont stockées dans les mémoires RAM (« *Random Access Memory* » en anglais) des serveurs informatiques formant le système informatique en nuage.

Le SSD-Cloud est un système informatique en nuage dont les données sont stockées dans des disques SSD (« *Solid State Drive* » en anglais).

Ainsi, les mémoires RAM ou les disques SSD permettent au procédé d'échange de données d'accéder ou d'écrire des données dans des temps très brefs.

*****

Dans des modes de réalisation particuliers de l'invention, la paire comprend au moins une information de routage liée au client informatique, et peut être notée

```
{     clé k      = (type_paire, nom_table, clé_primaire) ;
      valeur v   = (objet x)
},
```

où :

- La clé k composite de la paire est une chaîne de caractères contenant au moins les informations (type_paire, nom_table, clé_primaire), où :
    - L'information type_paire est une constante, par exemple « table », indiquant que la paire contient les informations relatives à une table, par opposition aux informations relatives à un index ou à un autre type ;
    - L'information nom_table est le nom « métier » attribué à la table. Ce nom intégré dans la clé de la paire, permet d'éviter les collisions entre les paires de tables différentes.
      Par exemple, un nom de table peut être « routes courantes » ou « HR » (pour l'anglais « *hot route* ») ;
    - L'information clé_primaire est la clé « métier » utilisée pour identifier la donnée de la paire de façon unique dans la table nom_table.
      Par exemple, pour une table nommée « routes courantes », la clé primaire peut être une adresse IP ;
- La valeur v de la paire contient un objet x quelconque fourni par l'application, puis sérialisé et éventuellement compressé et/ou chiffré à la volée lors de l'écriture de la paire, les actions inverses étant réalisées à la volée lors de la lecture ultérieure de la paire. Les données peuvent être par exemple structurées sous la forme d'une liste, d'une table, ou de toute combinaison de structures de données que l'application de routage sait gérer.

*****

Dans des modes de réalisation particuliers de l'invention, la clé d'identification de la paire associée à un client informatique correspond à l'identifiant du client informatique ou à la version du logiciel du client informatique.

*****

Dans des modes de réalisation particuliers de l'invention, le moteur de base de données est un moteur de type KVS (« *Key-Value Store* » en anglais) c.-à-d. un moteur capable de stocker des paires { clé ; valeur } .

Ainsi, le moteur est versatile, c'est-à-dire souple d'utilisation. En outre, le moteur est adapté à gérer un important volume de données car il peut maintenir ses fonctionnalités et ses performances en cas de forte demande.

*****

Dans des modes de réalisation particuliers de l'invention, le serveur de données stocke R réplicas de la paire, où R est une constante configurable entière au moins égale à trois.

Ainsi, les réplicas étant des exemplaires identiques, les données sont stockées de manière redondante. Il est alors possible de vérifier si les données stockées sont récentes ou obsolètes.

Il convient de noter que le nombre de réplica R est avantageusement impair afin de déterminer une majorité supérieure à la moitié. Dans le cas où le nombre de copies est pair, la majorité est acquise lorsque le nombre atteint la moitié plus un des copies.

*****

Dans des modes de réalisation particuliers de l'invention, chaque réplica est stocké dans une base de données distincte.

Ainsi, l'accès aux données redondantes stockées dans des bases différentes est plus rapide.

*****

Dans des modes de réalisation particuliers de l'invention, chaque nœud informatique comprend au moins un moteur de base de données, chaque moteur gérant au moins une base de données.

Ainsi, plusieurs moteurs de base de données sont présents dans le système informatique en nuage. L'accès aux données redondantes peut ainsi être concomitant. Les bases de données peuvent être sur le nœud informatique local au moteur de base de données ou sur un autre nœud informatique.

*****

Dans des modes de réalisation particuliers de l'invention, chaque base de données gérée par un moteur de base de données est hébergée localement sur le nœud informatique dudit moteur.

Ainsi, les bases étant avantageusement sur un nœud local au moteur, l'accès à chaque base est plus rapide.

*****

Dans des modes de réalisation particuliers de l'invention, chaque nœud informatique correspond à un seul serveur de routage distinct, chaque nœud informatique comprenant un seul moteur de base de données gérant une seule base de données hébergée sur ledit nœud informatique.

Il convient de noter qu'un nœud informatique mutualisant ainsi une capacité de routage plus un moteur et une base de données, peut également être appelé nœud mutualisé ou serveur mutualisé.

Ainsi, dans ce fonctionnement optimal comportant une pluralité de nœuds mutualisés dont chacun héberge une ou des instances de l'application de routage plus un moteur de base de données plus une base de données, l'accès aux informations de routage est très rapide. En utilisant des technologies connues de l'homme du métier, telles que Infiniband / RDMA (« Remote Direct Access Memory » en anglais) ou HSE / RoCE (« High Speed Ethernet » / « RDMA over Converged Enhanced Ethernet » en anglais), sur les serveurs du système informatique en nuage, il est possible d'obtenir des temps de réponse en lecture d'un réplica sur une base de données distante de l'ordre de dix microsecondes voire moins.

*****

Dans des modes de réalisation particuliers de l'invention, les moyens de stockage de données hébergent une table d'index regroupant au moins une partie des clés des paires stockées dans le système informatique en nuage, les clés des paires étant enregistrées dans les données d'une paire d'index notée { index_cle ; index_valeur }.

Ainsi, tout ou partie des données stockées dans la ou les bases de données sont accessibles à partir des clés stockées dans la table d'index sans connaître une clé en particulier.

*****

Dans des modes de réalisation particuliers de l'invention, la table d'index est divisée en F fragments d'index où F est une constante configurable entière positive non nulle, et une fonction de hachage déterministe répartit uniformément les clés des paires entre les fragments d'index.

En d'autres termes, la table d'index est constituée de plusieurs fragments d'index, pouvant également être appelés compartiments ou groupes. Chaque fragment est associé à une paire d'index notée

```
{    index_clé  = (type_paire, nom_table, num_fragment, hôte_client) ;
   Index_valeur  = (liste_clés)
},
```

où :

- La clé composite index_clé est une chaîne de caractères comprenant au moins les informations (type_paire, nom_table, num_fragment, hôte_client), où :
    - L'information type_paire est une constante, par exemple « index », indiquant que la paire contient les informations relatives à un index, par opposition aux informations relatives à une table ou à un autre type ;
    - L'information nom_table est le nom « métier » attribué à la table.
      Par exemple, un nom de table peut être « routes courantes » ou « HR » ;
    - L'information num_fragment est le numéro du fragment de l'index, nombre entier compris dans l'intervalle [1, F] ;
    - L'information hôte_client est le nom d'hôte (« hostname » en anglais) du serveur de routage depuis lequel une instance de l'application informatique effectue une demande d'indexation pour une clé ;
- La valeur index_valeur contient la liste liste_clés d'une partie des clés primaires des paires de la table nom_table, pour chaque clé ayant fait l'objet d'une indexation. Une fonction de hachage permet de répartir les clés entre les fragments.

Ainsi, l'ajout d'une clé dans la table d'index est plus rapide puisque circonscrit à un seul fragment de l'index, déterminé par une fonction de hachage déterministe sur la clé à ajouter entre les différents fragments de la table d'index.

D'autre part, le procédé d'échange de données comprenant une table d'index découpée en fragments est avantageusement agnostique, c'est-à-dire qu'il peut s'adapter aux composants informatiques utilisés, en termes de type et de nombre. En effet, le procédé d'échange peut ainsi gérer le stockage des paires d'index sur des moteurs KVS existant sur le marché possédant des limitations quant au volume maximum de données qu'une paire peut contenir.

*****

Dans des modes de réalisation particuliers de l'invention, la table d'index comprend un plan d'index propre à chaque serveur de routage, chaque plan d'index étant divisé en F fragments d'index, chaque plan d'index regroupant les clés des paires ayant déjà fait l'objet d'une indexation à la demande d'une instance de l'application opérée sur le serveur de routage associé audit plan d'index.

Ainsi, la table d'index comprend plusieurs plans d'index, permettant une indexation dite multiplanaire. Chaque plan d'index regroupe les fragments d'index relatifs à un seul serveur de routage. La présence de l'information hôte_client dans la clé composite notée index_clé = (type_paire, nom_table, num_fragment, hôte_client) associée au fragment d'index, a pour effet que les instances de l'application de routage opérées sur un serveur de routage, écrivent dans les paires des fragments de l'index qui sont propres à ce serveur de routage. En d'autres termes, une

table d'index est hébergée dans le nuage et comprend plusieurs plans dont chacun est propre à un serveur client du nuage, chaque plan étant lui-même découpé en fragments.

*****

Dans des modes de réalisation particuliers de l'invention, un verrou est associé à chaque fragment d'index de chaque plan d'index, ledit verrou autorisant une seule écriture simultanée dans ledit fragment d'index associé audit verrou, par les instances de l'application de routage opérées sur ledit serveur de routage.

Ainsi, il n'est possible de réaliser qu'une seule écriture à la fois pour chaque fragment d'un plan d'index de la table d'index. Il convient de noter que l'écriture d'une clé dans un plan d'index est effectuée par une instance de l'application opérée sur le serveur de routage associé au plan d'index.

Ainsi, lorsque deux instances i1 et i2 de l'application de routage demandent soit l'indexation d'une même clé, soit l'indexation de deux clés distinctes telles que la fonction de hachage les associe à un même fragment, alors :

- Si les deux instances i1 et respectivement i2 sont opérées sur deux serveurs distincts host_a et respectivement host_b, alors l'ajout de la clé se fait dans le même fragment de l'index mais dans deux paires différentes dont la clé vaut index_clé = (type_paire, nom_table, num_fragment, "host_a") pour l'instance i1 et index_clé = (type_paire, nom_table, num_fragment, "host_b") pour l'instance i2. Les deux instances écrivant dans deux plans distincts de l'index, il n'existe pas de concurrence d'accès donc aucun verrou n'est nécessaire pour départager les deux instances ;
- Si les deux instances i1 et i2 sont opérées sur un même serveur de routage host_a, alors l'ajout de la clé se fait dans la même paire du même fragment de l'index dont la clé vaut index_clé = (type_paire, nom_table, num_fragment, "host_a"). Il existe une concurrence d'accès à la paire du fragment entre les deux instances, arbitrée par un verrou localisé sur le serveur de routage host_a et associé au triplet (type_paire, nom_table, num_fragment). Par exemple, ce verrou peut être positionné sur un fichier localisé sur host_a et dont le nom comporte au moins les informations du triplet (type_paire, nom_table, num_fragment). Le nom du fichier peut être par exemple égal à la chaîne de caractères « table_HR_2.lock ».

Par ailleurs, la lecture de la totalité de l'index, qui consiste, à partir de n'importe quel serveur de routage, à lire tous les plans d'index pour tous les fragments d'index puis à fusionner et dé-doublonner les listes des clés primaires indexées, s'effectue également sans concurrence entre les serveurs de routage. Grâce à ce procédé d'index multiplanaire, les opérations d'écritures dans la table d'index (indexation donc ajout d'une clé dans un fragment) comme les opérations de lecture depuis un index (lecture de toutes les clés) s'exécutent sans concurrence donc sans contention entre des instances de l'application de routage opérées sur des serveurs de routage distincts. Ceci est cohérent avec la meilleure performance du système d'échange de données et une capacité de mise à l'échelle horizontale puisque l'augmentation du nombre de serveurs de routage ne provoque pas d'augmentation de la contention.

*****

Dans des modes de réalisation particuliers de l'invention, le procédé d'échange de données comprend une étape de nettoyage de la table d'index, ladite étape retirant une partie des paires liées à des clés obsolètes, ladite étape de nettoyage étant réalisée dans un temps maximum défini par l'instance demandant un nettoyage de la table d'index.

Ainsi, le temps de nettoyage étant garanti ne ralentit pas le procédé d'échange. En d'autres termes, le nettoyage peut être effectué en temps masqué. Le nettoyage de la table d'index, une

fois interrompu par ce temps garanti, peut être repris à tout instant, notamment lorsque la charge de requête sur la table d'index est plus faible.

*****

Dans des modes de réalisation particuliers de l'invention, une liste des bases de données dans lesquelles sont stockées chaque réplica d'une paire, dite liste des réplicas, est déterminée par l'intermédiaire d'une fonction de hachage à partir de la clé de la paire.

Ainsi, les copies des données stockées dans la base de données sont réparties entre différentes bases par une fonction de hachage.

Il convient de noter que les réplicas d'une paire peuvent être avantageusement hébergés sur des nœuds informatiques distincts.

*****

Dans des modes de réalisation particuliers de l'invention, la fonction de hachage entre les nœuds est déterministe, uniforme et consistante.

Ainsi, la fonction de hachage répartit uniformément et de manière déterministe les paires sur l'ensemble des bases de données. En outre, le hachage est consistant, également appelé hachage cohérent, c'est-à-dire que lorsque la taille de la table de hachage change, correspondant ainsi à un ajout ou à une suppression d'une base de données, alors le nombre de clés devant être redistribuées est égal à K/N, où K correspond au nombre de clés et N au nombre de bases, ce qui correspond au meilleur possible prévu par la théorie.

A partir d'une clé, le hachage désigne une base de données, appelée premier réplica. La liste des autres réplicas est ensuite déterminée à partir du premier réplica en prenant par exemple les bases suivantes dans la liste préconfigurée de toutes les bases existantes.

*****

Dans des modes de réalisation particuliers de l'invention, un procédé dit d'écriture consolidée permet à l'application cliente du nuage invoquant ledit procédé, d'écrire une paire notée { clé k ; valeur v = (objet x) } sur plusieurs réplicas.

Il convient de noter qu'une instance de l'application cliente du nuage invoquant une écriture consolidée est appelée l'invoquant, et que le nœud depuis lequel ladite instance effectue cette invocation est appelé le nœud invoquant.

*****

Dans des modes de réalisation particuliers de l'invention, l'écriture consolidée d'une paire s'effectue selon un procédé d'écriture comprenant les étapes suivantes :

1. Détermination d'une liste de réplicas à partir de la clé par l'intermédiaire de la fonction de hachage ;
2. Écriture de l'information sous la forme d'une paire copiée à l'identique sur chaque réplica.

*****

Dans des modes de réalisation particuliers de l'invention, l'écriture consolidée d'une paire notée { clé k ; valeur v = (objet x) } insère un horodatage dans la paire notée { clé k ; valeur v = (objet x, horodatage h) } copiée à l'identique sur chaque réplica.

Ainsi, chaque copie d'une paire comprenant normalement un horodatage identique, il est possible de déterminer quelles sont les copies qui sont obsolètes.

*****

Dans des modes de réalisation particuliers de l'invention, l'écriture consolidée d'une paire s'effectue d'abord sur une base de données hébergée sur le nœud local où est opéré le procédé d'écriture si ladite base est comprise dans la liste des réplicas.

Ainsi, la première écriture réussie peut être atteinte rapidement.

*****

Dans des modes de réalisation particuliers de l'invention, l'écriture consolidée d'une paire s'effectue en deux temps :

1. Écriture synchrone de la paire sur les réplicas, jusqu'à atteindre un quorum d'écritures réussies ;
2. Lorsque le quorum d'écritures réussies est atteint, écriture asynchrone de la paire sur les autres réplicas.

Ainsi, le temps de traitement de l'écriture consolidée d'une paire est rapide car il correspond à l'écriture réussie sur un quorum des réplicas.

*****

Dans des modes de réalisation particuliers de l'invention, le quorum d'écritures est strictement supérieur à la moitié du nombre de réplicas.

Ainsi, le quorum d'écriture, noté Q, correspond à la majorité des réplicas.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur de type KVS et une base de données qui contribuent au nuage, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un **procédé de lecture consolidée** d'une paire également appelé **procédé LC1**.

Il convient de noter qu'une instance de l'application cliente du nuage invoquant une lecture consolidée est appelée l'invoquant, et que le nœud depuis lequel ladite instance effectue cette invocation est appelé le nœud invoquant.

Ce procédé de lecture consolidée permet de reconstituer la paire d'origine c.-à-d. la paire notée { clé k ; valeur v = (objet x) } dont l'écriture consolidée sur plusieurs réplicas a été réalisée antérieurement.

Ce procédé de lecture consolidée d'une paire dans le nuage s'effectue en cinq temps :

1. Détermination d'une liste de réplicas à partir de la clé k via la même fonction de hachage que celle utilisée par le procédé d'écriture consolidée ;
2. Lecture de tout ou partie des réplicas notés { clé k ; valeur v = (objet x, horodatage h) } ;
3. Sélection de la paire réponse correspondant au réplica lu ayant la meilleure probabilité d'être identique à la paire d'origine ;
4. Suppression, dans la paire réponse, de l'horodatage h qui avait été inséré lors de l'écriture consolidée antérieure de la paire d'origine ;
5. Transmission à l'invoquant, de la paire réponse notée { clé k ; valeur v = (objet x) }.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur et une base de données qui contribuent au nuage, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un procédé nommé **condition technique** { en erreur ; non en erreur } d'une base de données du nuage.

La condition technique { en erreur ; non en erreur } de la base de données hébergée sur un nœud s, également appelée condition technique de la base s ou du nœud s ou du nœud/base s ou de s, est égale au **statut** { en erreur ; non en erreur } de la plus récente requête en lecture ou écriture d'un réplica dans la base s, exécutée à la demande de l'application cliente du nuage.

L'application cliente du nuage étant multi-instanciée sur une pluralité de nœuds, le procédé d'accès au nuage est identiquement multi-instancié sur la même pluralité de nœuds. Les différentes instances du procédé d'accès opérées sur un nœud n évaluent de manière unique et partagée la condition technique d'un nœud s, s pouvant être égal à n puisque les nœuds sont mutualisés. Les différentes instances du procédé d'accès opérées sur des nœuds distincts

évaluent distinctement la condition technique d'un nœud s. En d'autres termes, la condition technique d'un nœud s est toujours évaluée du point de vue du nœud invoquant n.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur et une base de données qui contribuent au nuage, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un **procédé hystérétique** déterminant un **état hystérétique** ou **état** { disponible ; indisponible } d'une base de données du nuage.

L'état hystérétique { disponible ; indisponible } de la base de données hébergée sur un nœud s, également appelé état hystérétique de la base s ou du nœud s ou du nœud/base s ou de s, est déterminé par l'intermédiaire d'un compteur d'erreurs fonctionnant en cycle d'hystérésis alimenté par la condition technique de s.

L'application cliente du nuage étant multi-instanciée sur une pluralité de nœuds, le procédé d'accès au nuage est identiquement multi-instancié sur la même pluralité de nœuds. Les différentes instances du procédé d'accès opérées sur un nœud n évaluent de manière unique et partagée l'état hystérétique d'un nœud s, s pouvant être égal à n puisque les nœuds sont mutualisés. Les différentes instances du procédé d'accès opérées sur des nœuds distincts évaluent distinctement l'état hystérétique d'un nœud s. En d'autres termes, l'état hystérétique d'un nœud s est toujours évalué du point de vue du nœud invoquant n.

Pour chaque requête en lecture ou écriture d'un réplica sur un nœud s, invoquée par une instance de l'application cliente du nuage opérée sur un nœud invoquant, le procédé hystérétique opéré sur le nœud invoquant observe le statut { en erreur ; non en erreur } de la requête.

Si le statut de la requête est en erreur alors la condition technique de s est en erreur, du point de vue du nœud invoquant. Si le statut de la requête est non en erreur alors la condition technique de s est non en erreur, du point de vue du nœud invoquant.

Selon que la condition technique de s est en erreur ou respectivement non en erreur du point de vue du nœud invoquant, un compteur d'erreurs noté c.s associé à s et opéré sur le nœud invoquant, est incrémenté ou respectivement décrémenté d'une unité. La valeur dudit compteur est bornée sur l'intervalle [0, E] où E est une constante configurable entière positive non nulle correspondant à un nombre maximum d'erreurs.

Cette variable c.s de type compteur d'erreurs permet de déterminer deux autres variables e.s et i.s associées à s et opérées sur le nœud invoquant, où e.s correspond à l'état hystérétique { disponible ; indisponible } de s et i.s correspond à l'horodatage de la dernière inversion dudit état hystérétique de s, grâce au procédé suivant :

- Lorsque le compteur c.s est incrémenté de la valeur E-1 à E et si l'état e.s est égal à « disponible », alors e.s reçoit la valeur « indisponible » et i.s reçoit l'horodatage de maintenant c.-à-d. la date et l'heure courantes ;
- Lorsque le compteur c.s est décrémenté de la valeur 1 à 0 et si l'état e.s est égal à « indisponible », alors e.s reçoit la valeur « disponible » et i.s reçoit l'horodatage de maintenant c.-à-d. la date et l'heure courantes ;
- Dans les autres cas, les variables e.s et i.s demeurent inchangées.

Il convient de noter que ces trois variables (c.s, e.s , i.s) associées à s et opérées par le procédé hystérétique sur le nœud invoquant, sont persistantes sur le nœud invoquant entre les requêtes.

Chaque fois que l'état e.s change c.-à-d. qu'il s'inverse, le triplet (s, e.s, i.s) est écrit dans une table nommée **TableEtats** opérée sur le nœud invoquant. Ladite table contient au plus un triplet (s, e.s, i.s) pour chaque valeur de s possible. L'opération d'écriture consiste soit à insérer

dans la table le triplet (s, e.s, i.s) si la table ne contient pas déjà un triplet associé à s, soit dans le cas contraire à mettre à jour dans la table les informations e.s et i.s dans le triplet existant associé à s. Des nœuds distincts opérant l'application cliente du nuage possèdent chacun leur propre table TableEtats.

Ce procédé est avantageux. En effet, certaines erreurs occasionnelles et non durables (perte de paquet réseau, microcoupure, ralentissement passager du système d'exploitation sur les nœuds, etc.) se produisant lors des accès à une base peuvent constituer du bruit indésirable si le problème est transitoire et ne correspond pas à un incident sérieux.

Il convient de noter que la sensibilité de l'état hystérétique à ce bruit est corrélée négativement à la valeur de E. Inversement, le délai de décision de l'état hystérétique d'un nœud est corrélé positivement à la valeur de E. Aux extrêmes, une valeur minimale (E=1) force l'état hystérétique égal à la condition technique, tandis qu'une valeur infinie (E=∞) empêche de décider l'état hystérétique.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur et une base de données qui contribuent au nuage, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un procédé nommé **période de rémanence** ou **rémanence** d'une base de données du nuage.

La période de rémanence de la base de données hébergée sur un nœud s, également appelée rémanence de la base s ou du nœud s ou du nœud/base s ou de s, est déterminée à partir de l'état hystérétique de s et de l'horodatage de la dernière inversion dudit état hystérétique de s.

L'application cliente du nuage étant multi-instanciée sur une pluralité de nœuds, le procédé d'accès au nuage est identiquement multi-instancié sur la même pluralité de nœuds. Les différentes instances du procédé d'accès opérées sur un nœud n évaluent de manière unique et partagée la rémanence d'un nœud s, s pouvant être égal à n puisque les nœuds sont mutualisés. Les différentes instances du procédé d'accès opérées sur des nœuds distincts, évaluent distinctement la rémanence d'un nœud s. En d'autres termes, la rémanence d'un nœud s est toujours évaluée du point de vue du nœud invoquant n.

La rémanence d'un nœud s du point de vue du nœud invoquant est déterminée à partir des variables (e.s, i.s) alimentées par le procédé hystérétique opéré sur le nœud invoquant.

La rémanence d'un nœud s du point de vue du nœud invoquant débute à l'instant où l'état e.s hystérétique du nœud s du point de vue du nœud invoquant évolue de « indisponible » vers « disponible » et se termine après une durée D, constante configurable positive ou nulle. Une base de données ne peut pas être en rémanence lorsque D est nul.

En d'autres termes :

- Si l'état e.s hystérétique de s est « indisponible » du point de vue du nœud invoquant alors s n'est pas en période de rémanence du point de vue du nœud invoquant ;
- Si l'état e.s hystérétique de s est « disponible » du point de vue du nœud invoquant alors s est en période de rémanence du point de vue du nœud invoquant mais seulement si ledit état est constaté « disponible » depuis un délai inférieur à D.

Ce procédé est avantageux. En effet, pendant l'indisponibilité d'une base, un réplica peut disparaitre de ladite base (par exemple effacement de la mémoire conséquemment au redémarrage d'un moteur de type IMDBMS) ou manquer une création lorsque ladite base n'est plus accessible en écriture (par exemple problème réseau). Une fois la base redevenue disponible, une action de lecture dudit réplica sur ladite base ne produit pas d'erreur mais génère à tort une réponse négative (clé inexistante) appelée faux négatif. En d'autres termes, une

réponse négative résultant de la lecture d'un réplica sur une base en rémanence, possède une probabilité non acceptable d'être un faux négatif. Ce procédé permet d'ignorer les réponses négatives qui se produisent pendant la période de rémanence, donc de se prémunir contre ces faux négatifs, en attendant que les réplicas soient régénérés par les écritures consolidées des paires invoquées par l'application utilisatrice du nuage.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur de type KVS et une base de données qui contribuent au nuage, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un **procédé hystérétique de lecture consolidée** d'une paire, ou **procédé LC2**, lequel reprend le procédé LC1 en y intégrant les procédés de condition technique, d'état hystérétique, de table TableEtats, de rémanence.

La condition technique mesurée sur le flux réel des requêtes en lecture des réplicas alimente le procédé hystérétique, qui à son tour alimente la table TableEtats, laquelle fournit l'information qui permet de déterminer la période de rémanence superposée à l'état hystérétique. Cette association est triplement avantageuse :

- Elle réduit la sensibilité au bruit que constituent les erreurs ponctuelles lors des lectures de réplicas, ce qui épargne des changements d'états intempestifs qui autrement réduiraient la performance et la résilience ;
- Elle réduit le risque de faux négatifs, ce qui augmente la fiabilité ;
- Elle décide rapidement de l'état hystérétique, ce qui contribue à la performance et à la fiabilité et fonctionne en cercle vertueux. En effet, plus la sollicitation du nuage donc de chaque base est intensive par l'application cliente du nuage, plus l'état hystérétique d'une base est décidé rapidement. Tel ne serait pas le cas avec un procédé de supervision cyclique conventionnel.

Il convient de noter que les constantes configurables E et D influencent le comportement du procédé LC2. A titre d'exemple, lorsque la constante E est égale à trois erreurs et la constante D égale à soixante secondes, une base recevant dix-mille requêtes par seconde :

- Voit son état hystérétique décidé en un temps égal à seulement trois cent microsecondes (E/10000) ;
- Est insensible à deux erreurs intempestives consécutives (E-1) ;
- Reste protégée contre les faux négatifs pendant une période de rémanence de soixante secondes (D) après une indisponibilité révolue.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur de type KVS et une base de données qui contribuent au nuage, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un **procédé hystérétique probabiliste rétroactif de lecture consolidée** d'une paire, ou **procédé LC3**, lequel reprend le procédé LC2 en y intégrant un procédé nommé **LoiTransition** décrit ci-après.

Dans ce mode particulier, la lecture d'un réplica sur un nœud n'est exécutée qu'avec une certaine probabilité, laquelle est déterminée par une fonction de probabilité nommée LoiTransition. Avant chaque lecture envisagée d'un réplica sur un nœud, un tirage aléatoire est effectué puis comparé à la probabilité déterminée par la fonction LoiTransition. Si le tirage aléatoire est inférieur ou égal à la valeur déterminée par la fonction LoiTransition, alors le réplica est lu pour être pris en compte par le procédé de lecture consolidée de la paire, sinon il est ignoré et n'est pas contributif au résultat de la lecture consolidée.

Il convient de noter que la fonction LoiTransition ne s'applique qu'aux requêtes en lecture. En effet, les requêtes en écriture n'ont pas besoin d'être régulées car elles sont rares comparativement aux lectures – par hypothèse d'usage d'un moteur de type NoSQL.

Lorsque l'état hystérétique d'un nœud s'inverse de « disponible » vers « indisponible », le procédé LC3 cesse progressivement d'envoyer les requêtes vers ledit nœud afin de le décharger des sollicitations en lecture de façon linéairement décroissante sur une période d'amortissement de durée A, où A est une constante configurable exprimant une durée positive non nulle.

Cette baisse progressive du flux de requêtes est avantageuse pour deux raisons :

- Peu de temps après que l'état hystérétique d'un nœud devienne « indisponible », la proportion des requêtes en lecture continuant à être transmises audit nœud est encore importante puisque la décroissance est linéaire, ce qui garantit une grande réactivité du cycle d'hystérésis donc une transition rapide vers l'état « disponible » si l'incident affectant ledit nœud est de courte durée ;
- À l'inverse, si un nœud est dans l'état « indisponible » depuis longtemps, la proportion des requêtes en lecture continuant à être transmises audit nœud est faible. Dans ce cas, la transition vers l'état « disponible » demande un délai plus important, mais sans que cela constitue un inconvénient puisque ledit délai reste négligeable relativement à la durée déjà importante de l'incident.

  En d'autres termes, le délai de réactivité de l'état hystérétique est corrélé positivement à la durée de l'incident, ce qui lui permet de rester en toute circonstance relativement négligeable.

Passé ce délai A, la sollicitation est maintenue à un plancher fixé arbitrairement à 1%, ce seuil étant configurable. Ce flux non nul de requêtes est avantageux car il permet d'alimenter le fonctionnement du procédé hystérétique en toute circonstance, ce qui dispense de mettre en place un système de supervision.

Symétriquement, lorsque l'état hystérétique d'un nœud s'inverse de « indisponible » vers « disponible », le procédé LC3 recommence progressivement à envoyer les requêtes vers ledit nœud de façon linéairement croissante sur une période A. Cette hausse progressive est avantageuse car elle permet de réduire le risque d'incident inhérent à une remise en charge brutale.

Passé ce délai A de remontée en charge, la sollicitation revient à son niveau nominale de 100%.

De manière plus précise, cette fonction LoiTransition détermine la probabilité p de lire un réplica sur un nœud, en fonction de l'état e hystérétique dudit nœud, du délai d écoulé depuis la dernière inversion d'état hystérétique dudit nœud, et de la constante d'amortissement A, où :

- Le délai d se calcule comme l'horodatage de maintenant moins l'horodatage i de la dernière inversion d'état hystérétique du nœud ;
- L'état e hystérétique et l'horodatage i de la dernière inversion d'état pour un nœud donné proviennent de la table TableEtats elle-même alimentée par le procédé hystérétique.

Plus précisément encore, la fonction de probabilité LoiTransition est pleinement définie ainsi :

- Si l'état e hystérétique est « indisponible » et si d $\in$ [0, A] alors :

$$p = 1 - (1 - 0.01) * \frac{d}{A} = 1 - \frac{0.99 * d}{A}$$

- Si l'état e hystérétique est « indisponible » et si d $\in$]A, +∞[alors :

$$p = 0.01\ (1\%)$$

- Si l'état e hystérétique est « disponible » et si d $\in$ [0, A] alors :

$$p = 0.01 + (1 - 0.01) * \frac{d}{A} = 0.01 + \frac{0.99 * d}{A}$$

- Si l'état e hystérétique est « disponible » et si d $\in$]A, +∞[alors :

$$p = 1\ (100\%)$$

La probabilité p peut également s'écrire p = LoiTransition(e, d, A) où (e, d, A) sont les trois paramètres de la fonction LoiTransition.

Il convient de noter que ce procédé LC3 est dit rétroactif car il comporte une rétroaction temporelle. En effet, la lecture des réplicas alimente le procédé hystérétique qui alimente à son tour la table TableEtats, laquelle alimente d'une part le calcul de la rémanence et d'autre part les paramètres transmis à la fonction de probabilité LoiTransition, les deux ayant pour effet de modifier la cinématique de lecture des réplicas lus, et ainsi de suite.

*****

Dans des modes de réalisation particuliers de l'invention comprenant une pluralité de nœuds mutualisés dont chacun opère une ou des instances de l'application cliente du nuage plus un moteur de type KVS IMDBMS et une base de données qui contribuent au nuage de type RAM-Cloud, chaque instance de l'application cliente du nuage utilise un procédé d'accès au nuage comprenant un **procédé hystérétique probabiliste rétroactif intégré de lecture consolidée** d'une paire, ou **procédé intégré LC4**, lequel reprend le procédé LC3 en y intégrant les étapes suivantes :

- La **paramétrisation LC4**, qui consiste, pour un invoquant c.-à-d. pour une instance de l'application invoquant la lecture consolidée d'une paire dans le nuage, à transmettre au prologue LC4 les paramètres associés à la demande ;
- La **fonction ou procédé de lecture consolidée LC4**, comprenant :
  - Le **prologue LC4**, qui consiste à recevoir les paramètres transmis par la paramétrisation LC4 puis à préparer l'exécution du corps LC4 ;
  - Le **corps LC4**, qui consiste à déterminer la paire réponse en optimisant la performance et la fiabilité, puis à retourner la paire réponse à l'invoquant.

Dans ce mode particulier, la **paramétrisation LC4** comprend la transmission d'un paramètre obligatoire et de deux paramètres facultatifs.

Primo, l'invoquant transmet obligatoirement un paramètre nommé k correspondant à la clé de la paire dont la lecture consolidée est invoquée.

Secundo, l'invoquant peut en option transmettre un paramètre nommé full_scan ou f, de type booléen { VRAI ; FAUX }, qui permet de maximiser la fiabilité de la lecture consolidée d'une paire dans le nuage. Par défaut, ce paramètre est égal à FAUX c.-à-d. qu'il est privé d'effet. Lorsque ce paramètre est transmis égal à VRAI, alors il affecte le comportement du corps LC4 de la manière suivante :

- Tous les réplicas de la paire sont lus, sans possibilité de se limiter à un sous ensemble ;
- Si tous les réplicas produisent une erreur ou un dépassement du temps autorisé lors de la lecture, alors une réponse caractérisant une erreur est retournée à l'invoquant ;
- Si aucun réplica n'existe, alors une réponse négative est retournée à l'invoquant, signifiant que la paire n'existe pas ;
- Si un ou plusieurs réplicas existent, alors le réplica dont l'horodatage est le plus récent est sélectionné comme paire réponse retournée à l'invoquant.

Ce paramètre full_scan permet avantageusement à l'invoquant de maximiser la fiabilité de la lecture consolidée LC4 d'une paire dans le nuage, en contrepartie d'une diminution de la performance.

Tertio, l'invoquant peut en option transmettre un paramètre nommé joker ou j, qui permet de maximiser la performance de la lecture consolidée LC4 d'une paire dans le nuage. Ce paramètre correspond à l'âge en dessous duquel un réplica peut être considéré comme étant suffisamment fiable pour être sélectionné comme paire réponse sans chercher à lire d'autres réplicas. Un joker de valeur nulle est privé d'effet car il ne peut exister un réplica d'âge négatif. Par défaut, ce paramètre est égal à zéro c.-à-d. qu'il est privé d'effet. Lorsque ce paramètre est positif et non nul, alors il affecte le comportement du corps LC4 de la manière suivante :

- A chaque lecture d'un réplica, son horodatage est utilisé pour calculer l'âge dudit réplica, correspondant au délai écoulé depuis la création ou la dernière mise à jour dudit réplica ;
- Si cet âge est inférieur à la valeur du joker, alors ledit réplica constitue la paire réponse sans qu'il soit nécessaire de lire d'autres réplicas ;
- Si aucun réplica ne remplit la condition d'âge, alors le joker n'a aucun effet.

Il convient de noter que l'augmentation de la performance est obtenue sans diminuer la fiabilité. En effet, perdre en fiabilité implique de sélectionner à tort une paire réponse, c.-à-d. de sélectionner comme paire réponse un réplica plus jeune que la valeur du joker alors qu'il existe un autre réplica plus récent, ce qui revient à réunir les deux conditions suivantes :

- Puisque le réplica sélectionné en tant que paire réponse n'est pas le plus récent, alors la valeur du paramètre joker est incorrecte, ce qui caractérise une erreur humaine de paramétrage ou de codage de l'application cliente du nuage ;
- L'existence d'au moins un réplica plus récent que celui sélectionné à tort implique que la paire a été mise à jour via une écriture consolidée mais que le réplica sélectionné à tort a raté cette mise à jour, ce qui caractérise une incohérence conséquemment à un dysfonctionnement technique.

La probabilité de réunir ces deux conditions étant infime, il en résulte que l'utilisation du paramètre joker ne dégrade pas la fiabilité. Ce paramètre joker permet avantageusement à l'application cliente du nuage de maximiser la performance de la lecture consolidée LC4 d'une paire dans le nuage, sans pour autant perdre en fiabilité.

Ces paramètres full_scan et joker s'excluent mutuellement et sont avantageux, car ils permettent à l'application cliente du nuage, qui a connaissance de la nature des données accédées, des caractéristiques individuelles de la paire accédée, et du contexte, d'influencer la performance et la fiabilité de la lecture consolidée LC4.

Dans ce mode particulier, le **prologue LC4** opère les actions ci-après :

1. Le prologue exécute les initialisations inconditionnelles suivantes :
    - Une variable notée r+ comptant les réponses positives des lectures de réplicas, est initialisée à zéro ;
    - Une variable notée r- comptant les réponses négatives des lectures de réplicas, est initialisée à zéro ;
    - Une variable notée x' correspondant à l'objet de référence, est initialisée à néant ;
    - Une variable notée h' correspondant à l'horodatage de référence c.-à-d. à l'horodatage de l'objet de référence, est initialisée à zéro ;
    - Une variable notée q contenant la valeur du quorum applicable pendant la lecture consolidée d'une paire, est initialisée égale à Q ;
    - Une variable notée m est initialisée à l'horodatage de maintenant c.-à-d. de la date et de l'heure courantes. Toutes les étapes du corps LC4 qui nécessitent de connaitre l'horodatage de maintenant se réfèrent à cette variable. Ainsi

l'horodatage du moment présent reste invariant pendant l'exécution d'une lecture consolidée ;

2. Puis, si le paramètre f est égal à VRAI, alors le prologue LC4 exécute les actions suivantes :
    - Le paramètre j, qu'il ait été ou non transmis via l'étape de paramétrisation LC4, est initialisé à zéro, ce qui privera d'effet le procédé du joker utilisé par le corps LC4 ;
    - La variable q est initialisée à l'infini, ce qui privera le corps LC4 de la possibilité d'atteindre un quorum et forcera ainsi la lecture de tous les réplicas ;
3. Puis une file ϕ définissant la liste des nœuds des réplicas et leur ordre prévisionnel de lecture par le corps LC4, est initialisée avec la liste des R nœuds contenant les R réplicas associés à la clé k, identifiés de façon déterministe à partir de la clé k par la même fonction de hachage que celle utilisée dans le procédé d'écriture consolidée. Cette file ϕ est ordonnée de manière aléatoire. Par exception, si le nœud invoquant appartient à la file, alors ce nœud local est positionné en tête de la file, de sorte qu'il sera accédé en premier par le corps LC4.
   Ce procédé est avantageux car il privilégie une première lecture locale. De plus, dans le cas où l'application utilise le paramètre joker, le procédé peut aboutir prématurément à un résultat consolidé en lisant un seul réplica, qui plus est local, qui plus est en mémoire puisque le moteur est de type IMDBMS. Cette conjonction est triplement performante sans pénaliser la fiabilité. Par exemple, si le nuage comprend quatre (N) nœuds et si le nombre de réplicas vaut trois (R), alors il existe une probabilité de 75% (N/R) qu'un nœud du nuage héberge l'un des réplicas d'une paire quelconque. Dans une telle configuration, toutes les requêtes de lectures consolidées s'effectuent en lisant un seul réplica (procédé du joker) en mémoire (moteur de type IMDBMS), soit sur un nœud local dans 75% des cas, soit sur un nœud distant dans 25% des cas.

Dans ce mode particulier, le **corps LC4** opère localement sur le nœud invoquant, les actions ci-après :

1er. Le corps, itérativement pour chaque nœud n prélevé en tête de la file ϕ jusqu'à épuisement de ladite file, opère les étapes suivantes :

A. Appliquer le procédé LoiTransition :
    a. Écrire dans une variable notée e.n l'état hystérétique du nœud n du point de vue du nœud invoquant. Cette information est préalablement lue dans la table TableEtats opérée localement sur le nœud invoquant ;
    b. Écrire dans une variable notée i.n l'horodatage de la dernière inversion d'état du nœud n du point de vue du nœud invoquant. Cette information est préalablement lue dans la table TableEtats opérée localement sur le nœud invoquant ;
    c. Écrire dans une variable notée d.n le délai écoulé depuis la dernière inversion d'état du nœud n du point de vue du nœud invoquant, calculé comme m moins i.n, où m est la variable initialisée lors du prologue LC4 ;
    d. Écrire dans une variable notée p.n la probabilité de lire le réplica sur le nœud n, calculée comme le résultat de la fonction LoiTransition appliquée aux trois paramètres (e.n, d.n, A) ;
    e. Effectuer un tirage aléatoire d'un nombre réel z sur l'intervalle [0, 1] ;
    f. Si ($z \leq p.n$) alors continuer au point B ci-après ;

Dans le cas contraire (z > p.n) alors ignorer le nœud n puis itérer le corps LC4 sur le prochain nœud dans la file ϕ ;

B. Lire le réplica associé à la clé k sur le nœud n, noté { clé k ; valeur v.n = (objet x.n, horodatage h.n) }, v.n étant la valeur lue associée à la clé k, x.n étant l'objet transmis par l'application lors de l'écriture consolidée, h.n étant l'horodatage inséré par l'écriture consolidée, puis opérer les actions suivantes :

a. Si la tentative de lecture du réplica sur le nœud n n'est pas terminée après une durée T, où T est une constante configurable positive non nulle, alors interrompre la tentative de lecture sur le nœud n puis itérer le corps LC4 sur le prochain nœud dans la file ϕ.
   Ce procédé garantit la durée de lecture d'un réplica, ce qui à son tour garantit la durée du procédé de lecture consolidée LC4 d'une paire, ce qui permet de traiter en temps réel les demandes provenant de l'application cliente du nuage ;
b. Si la tentative de lecture du réplica sur le nœud n produit une erreur, alors ignorer le nœud n puis itérer le corps LC4 sur le prochain nœud dans la file ϕ ;
c. Si la tentative de lecture du réplica sur le nœud n est négative c.-à-d. si la clé n'existe pas, alors :
   1. Si l'état e.n hystérétique du nœud est « disponible » et si (d.n < D) alors le nœud n est en rémanence du point de vue du nœud invoquant, donc la présente lecture négative sur n est considérée comme un faux négatif. En conséquence, ignorer le nœud n puis itérer le corps LC4 sur le prochain nœud dans la file ϕ ;
      Dans le cas contraire (le nœud n n'est pas en rémanence) continuer au point 2 ci-après ;
   2. Incrémenter d'une unité la variable r- comptant les réponses négatives ;
   3. Si (r- < q) alors le quorum des réponses négatives n'est pas atteint. En conséquence, itérer le corps LC4 sur le prochain nœud dans la file ϕ ;
      Dans le cas contraire (r- $\geq$ q) continuer au point 4 ci-après ;
   4. Le quorum q des réponses négatives est atteint. En conséquence, la lecture consolidée LC4 termine prématurément en statuant une réponse négative retournée à l'invoquant ;
d. Si la tentative de lecture du réplica sur le nœud n est positive alors :
   1. Si (j > 0) alors le procédé du joker est opérant. En conséquence, continuer au point 2 ci-après ;
      Dans le cas contraire (j = 0) continuer au point 3 ci-après ;
   2. Le procédé du joker est opérant et se déroule comme suit :
      i. Déterminer l'âge noté a.n du réplica lu sur le nœud n, calculé comme m moins h.n, où m est la variable initialisée lors du prologue LC4 ;
      ii. Si (a.n < j) alors le réplica lu sur le nœud n est plus jeune que la valeur du joker, et à ce titre réputé fiable. En conséquence, la lecture consolidée LC4 termine prématurément en statuant une réponse positive. La paire réponse notée { clé k ; valeur = (objet x.n) } retournée à l'invoquant correspond au réplica lu sur le nœud n, débarrassé de son horodatage ;

Dans le cas contraire (a.n ≥ j) continuer au point 3 ci-après ;

3. Si (h.n < h') c.-à-d. si l'horodatage du réplica lu sur le nœud n est inférieur à l'horodatage de référence, alors le réplica lu sur n est obsolète, ce qui constitue un cas d'incohérence des réplicas. En conséquence, ignorer le nœud n puis itérer le corps LC4 sur le prochain nœud dans la file ϕ ;
   Dans le cas contraire (h.n ≥ h') continuer au point 4 ci-après ;
4. Si (h.n > h') c.-à-d. si l'horodatage du réplica lu sur le nœud n est supérieur à l'horodatage de référence, alors le compteur r+ des réponses positives a comptabilisé des réplicas obsolètes lors des itérations précédentes sur les nœuds de la file ϕ, ce qui constitue un cas d'incohérence des réplicas. En conséquence continuer au point 5 ci-après ;
   Dans le cas contraire (h.n ≤ h') continuer au point 6 ci-après ;
5. Remettre à zéro le compteur r+ ;
6. Incrémenter d'une unité le compteur r+ des réponses positives ;
7. Copier l'objet x.n dans l'objet de référence x' et l'horodatage h.n dans l'horodatage de référence h' ;
8. Si (r+ < q) alors le quorum des réponses positives n'est pas atteint. En conséquence, itérer le corps LC4 sur le prochain nœud dans la file ϕ ;
   Dans le cas contraire (r+ ≥ q) continuer au point 9 ci-après ;
9. Le quorum des réponses positives est atteint. En conséquence, la lecture consolidée LC4 termine prématurément la requête en statuant une réponse positive. La paire réponse notée { clé k ; valeur = (objet x') } retournée à l'invoquant, comprend l'objet de référence x' c.-à-d. l'objet associé à l'horodatage de référence. En d'autres termes, la paire réponse est égale au réplica le plus récent parmi les réplicas lus, débarrassé de son horodatage ;

2e. Une fois la file ϕ épuisée et si le corps LC4 n'est pas parvenu à statuer prématurément une réponse par le procédé du joker ou par atteinte d'un quorum, alors :

A. Si (r+ > 0) alors il existe au moins une réponse positive. En conséquence, la lecture consolidée LC4 statue une réponse positive. La paire réponse notée { clé k ; valeur = (objet x') } retournée à l'invoquant, comprend l'objet de référence x' c.-à-d. l'objet associé à l'horodatage de référence. En d'autres termes, la paire réponse est égale au réplica le plus récent parmi les réplicas lus, débarrassé de son horodatage ;
   Dans le cas contraire (r+ = 0) continuer au point B ci-après ;
B. Si (r- > 0) alors il existe au moins une réponse négative. En conséquence, la lecture consolidée LC4 statue une réponse négative retournée à l'invoquant ;
   Dans le cas contraire (r- = 0) continuer au point C ci-après ;
C. Sinon, il n'existe aucune réponse positive ni négative c.-à-d. que tous les réplicas lus sont soit en erreur soit en dépassement du délai maximum T de lecture. En conséquence, la lecture consolidée LC4 statue une réponse en erreur retournée à l'invoquant.

*****

Dans des modes de réalisation particuliers de l'invention, le client informatique est un automate bancaire.

L'invention concerne également un produit programme d'ordinateur comprenant des instructions de code de programme pour l'exécution des étapes d'un procédé d'échange de données informatiques en temps réel entre un client informatique et un serveur informatique, ledit programme étant exécuté sur un ordinateur.

*****

Dans des modes de réalisation particuliers de l'invention, le produit programme d'ordinateur est inclus dans une bibliothèque logicielle, également appelé librairie.

Ainsi, l'installation est simple à réaliser et ne nécessite pas l'installation et l'exploitation d'un nouveau moteur de base de données et de son écosystème.

L'invention concerne également un système d'échange en temps réel de données informatiques comprenant une flotte de clients informatiques à l'origine des données informatiques, un parc de serveurs de traitement aptes à traiter lesdites données informatiques et des moyens de routage reliant les clients informatiques aux serveurs de traitement, les moyens de routage étant connectés à des moyens de stockage de données, les moyens de routage comprenant au moins un serveur de routage.

Selon l'invention, les moyens de stockage des données comprennent un système informatique en nuage stockant au moins une information de routage entre les clients informatiques et les serveurs de traitement dans une base de données, le système informatique en nuage comprenant une pluralité de nœuds informatiques, la base de données étant gérée par un moteur de base de données opérant sur un des nœuds informatiques.

*****

Dans des modes de réalisation particuliers de l'invention, le système informatique en nuage est un RAM-Cloud ou un SSD-Cloud.

*****

Dans des modes de réalisation particuliers de l'invention, la flotte de clients informatiques comprend des automates bancaires.

3.5 BRÈVE DESCRIPTION DES FIGURES

D'autres avantages, buts et caractéristiques particulières de la présente invention ressortiront de la description non limitative qui suit d'au moins un mode de réalisation particulier du dispositif et du procédé objets de la présente invention, en regard des dessins annexés, dans lesquels :

La Figure Formelle 1 représente un mode de réalisation d'un système d'échange en temps réel de données informatiques selon l'invention ;

La Figure Formelle 2 représente une variante du mode de réalisation en référence à la Figure Formelle 1, cette variante correspondant à un mode de réalisation optimal de l'invention ;

La Figure Formelle 3 est un schéma synoptique qui représente un procédé d'écriture consolidée (200) d'une paire dans les réplicas du système d'échange en temps réel de données informatiques ;

La Figure Formelle 4 et la Figure Formelle 5 sont deux moitiés d'un même schéma synoptique qui représente un procédé hystérétique probabiliste rétroactif intégré de lecture

consolidée (300) d'une paire dans les réplicas du système d'échange en temps réel de données informatiques.

Légende de la Figure Formelle 4 :

- ϕ : file des nœuds contenant les réplicas à lire ;
- a.n : âge de x.n ;
- d.n : délai écoulé depuis la dernière inversion d'état hystérétique du nœud n ;
- e.n : état hystérétique statué pour le nœud n ;
- f : paramètre « full_scan » (booléen) ;
- h.n : horodatage de x.n ;
- h' : horodatage de x' (horodatage de référence) ;
- i.n : horodatage de la dernière inversion d'état hystérétique du nœud n ;
- j : paramètre « joker » ;
- k : paramètre correspondant à la clé de la paire ;
- m : horodatage de maintenant (date et heure courantes) ;
- n : nœud courant correspondant à l'itération en cours sur ϕ ;
- p.n : probabilité de lecture d'un réplica sur le nœud n ;
- q : quorum applicable pendant la lecture consolidée ;
- r+ : compteur de réponses positives en itérant sur ϕ ;
- r- : compteur de réponses négatives en itérant sur ϕ ;
- v.n : valeur associée à k contenue dans le réplica lu sur le nœud n ;
- x.n : objet contenu dans v.n ;
- x' : objet de référence c.-à-d. meilleur objet x.n obtenu à ce stade du parcours de la file ϕ ;
- z : nombre réel aléatoire sur l'intervalle [0,1].

Légende de la Figure Formelle 5 :

- c.s : compteur d'erreurs associé au nœud s ;
- e.s : état hystérétique du nœud s ;
- i.s : horodatage de la dernière inversion d'état du nœud s ;
- s : nœud accédé.

3.6 DESCRIPTION DÉTAILLÉE DE MODES DE RÉALISATION DE L'INVENTION

La présente description est donnée à titre non limitatif, chaque caractéristique d'un mode de réalisation pouvant être combinée à toute autre caractéristique de tout autre mode de réalisation de manière avantageuse.

On note, dès à présent, que les figures ne sont pas à l'échelle.

3.7 EXEMPLE D'UN MODE DE RÉALISATION PARTICULIER DE L'INVENTION

Dans le présent exemple et sauf mention contraire, les constantes configurables sont valorisées ainsi :

- A est égale à trente secondes, correspondant à la durée d'amortissement lors des transitions de l'état hystérétique ;

- D est égale à soixante secondes, correspondant à la durée de la période de rémanence ;
- E est égale à trois erreurs, correspondant au plafond du compteur d'erreurs du procédé hystérétique ;
- F est égale à seize fragments, correspondant au nombre de fragments compris dans un index ;
- N est égale à quatre nœuds, correspondant au nombre de nœuds ;
- Q est égale à deux réplicas, correspondant au quorum pour les lecture et d'écriture consolidées ;
- R est égale à trois réplicas, correspondant au nombre de réplicas de toute paire ;
- T est égale à une milliseconde, correspondant à la durée maximum autorisée d'une lecture d'un réplica sur un nœud.

*****

La Figure Formelle 1 représente un système d'échange (100) de données informatiques. Le système d'échange de données comprend une flotte hétérogène de vingt-deux mille clients informatiques reliés à un parc de serveurs de traitement (120) des données informatiques par l'intermédiaire d'un dispositif de routage (130). Les clients informatiques sont dans le présent exemple non limitatif de l'invention des automates bancaires (110). Les serveurs de traitement (120), généralement hébergés dans un ou plusieurs centres de données couramment appelés « data-centers », sont connectés au réseau interbancaire (140).

Il convient de noter que les échanges de données bancaires sont effectués, dans le présent exemple non limitatif de l'invention, de manière sécurisée.

La flotte d'automates bancaires (110) comprend dans le présent exemple des distributeurs automatiques de billet (DAB) (111) et des guichets automatiques de banque (GAB) (112). Chaque automate (110) muni d'un écran d'affichage est en général situé dans le local d'une agence bancaire distincte dont le réseau s'étend sur un vaste territoire. Un automate (110) peut également être situé en dehors d'un local d'une agence bancaire, par exemple dans un complexe commercial ou dans une gare de transport.

Les automates bancaires (110) provenant de multiples constructeurs fonctionnent chacun avec un logiciel applicatif. Compte-tenu notamment des difficultés rencontrées pour déployer des mises à jour de logiciel sur l'ensemble de la flotte (105) d'automates (110), des versions différentes d'un même logiciel coexistent en permanence sur l'ensemble de la flotte.

Le dispositif de routage (130) comprend des routeurs physiques connectant les automates (110) aux serveurs de gestion (120), et quatre serveurs de routage (133). Les serveurs de routage (133) traitent chacun des milliers de processus également appelés instances d'une application informatique muti-instanciée qui permet de distribuer les requêtes entre les automates (110) et les serveurs de gestion (120).

Lorsqu'un utilisateur introduit, par exemple, sa carte bancaire dans un distributeur automatique de billet (111a) pour effectuer un retrait, l'utilisateur sélectionne l'opération qu'il souhaite effectuer, en l'occurrence ici un retrait d'un montant choisi par l'utilisateur, sur l'interface graphique s'affichant sur l'écran du distributeur automatique de billet (111a). Cette interface graphique est proposée par le logiciel métier du distributeur automatique (111a).

Le distributeur automatique (111a) effectue une demande d'ouverture d'une session sécurisée avec l'un quelconque des serveurs de gestion (120). Une requête provenant du distributeur automatique de billet (111a) transite alors par le système de routage vers un serveur de traitement (120) compatible avec le distributeur automatique de billet (111a). Le serveur de traitement (120) dialogue ensuite avec un serveur bancaire (141) du réseau interbancaire (140) afin d'autoriser ou non le retrait.

Au cours d'une session, le distributeur automatique (111a) émet de nombreuses requêtes, dont chacune comprend :

- Systématiquement, l'adresse IP atm_ip du distributeur automatique de billet (111a) ;
- Occasionnellement, l'identifiant atm_id du distributeur automatique de billet (111a) ;
- Occasionnellement, la version atm_version du logiciel du distributeur automatique de billet (111a) ;
- Systématiquement, l'horodatage req_time de la requête, précis au moins à la milliseconde, voire à la microseconde.

Il convient de noter que l'adresse IP du distributeur automatique (111a) change rarement. En tout état de cause, l'adresse IP du distributeur automatique (111a) reste constante tant que dure la session sécurisée avec un des serveurs de gestion (120). L'adresse IP atm_ip peut donc être utilisée comme identifiant temporaire de l'automate (111a) et permet de retrouver les informations atm_id ou atm_version de l'automate (111a) lorsque ces informations ne sont pas présentes dans une requête. En effet ces informations atm_id et atm_version peuvent être mémorisées dans la route courante associée à l'adresse IP atm_ip de l'automate.

La requête comprend une adresse URL (« *Uniform Resource Locator* » en anglais) dans laquelle sont contenues des informations sur le caractère terminal ou non terminal de la requête. En d'autres termes, l'adresse URL comprend des informations indiquant si la requête est liée ou non à la fin de la session. Dans le présent exemple, une fin de session peut correspondre par exemple au moment où l'utilisateur demande à retirer sa carte du distributeur (111a).

L'adresse URL comprend également des informations sur le caractère « PING » de la requête. Le « PING » émis par le distributeur automatique (111a) vers le serveur de gestion (120) traitant les requêtes du distributeur automatique (111a), permet au distributeur automatique (111a) de vérifier que la session est toujours ouverte avec le serveur de gestion (120).

Afin de distribuer quasi-instantanément l'ensemble des requêtes provenant des automates bancaires (110), les serveurs de routage (133) sont connectés à un système informatique en nuage de type RAM-Cloud (134) permettant de stocker des données.

Le RAM-Cloud (134) est un système informatique en nuage très efficace. En effet, le stockage des données informatiques dans le RAM-Cloud (134) est effectué dans les mémoires informatiques RAM (« *Random Access Memory* » en anglais) d'un ensemble de nœuds de réseau connectés les uns aux autres. Le temps de lecture ou d'écriture des données dans le RAM-Cloud (134) est très faible.

Le RAM-Cloud (134) décrit dans le présent exemple de réalisation non limitatif de l'invention, comprend quatre nœuds informatiques (135) correspondant chacun à un serveur informatique. Chacun des nœuds informatiques (135) comprend un ou plusieurs processeurs sur lesquels peuvent être opérées les fonctions d'une bibliothèque logicielle de stockage permettant de gérer le stockage de données. La bibliothèque logicielle de stockage comprend une fonction de lecture consolidée et une fonction d'écriture consolidée d'une paire dans le nuage.

Dans le présent exemple, chaque nœud (135) comprend un moteur de base de données de type KVS (« *Key Value Store* ») gérant une base de données (136) stockée localement sur ledit nœud (135).

Dans des variantes de ce mode de réalisation particulier de l'invention, un nœud peut héberger plusieurs moteurs de base de données de stockage de données de type KVS. Un nœud peut également ne pas héberger de serveur de stockage, auquel cas, le stockage des données sur le nœud est géré par un serveur de stockage hébergé sur un autre nœud.

Chaque base de données (136) stocke les données liées à une requête provenant d'un automate (110) dans une paire notée { clé k ; valeur v = (objet x) } où l'objet x contient les données. Les paires sont rassemblées en table de données en fonction de la nature des données

stockées dans chaque paire. La clé de la paire est une clé composite écrite sous la forme d'une chaîne de caractère, comprenant les informations (type_paire, nom_table, clé_primaire), où :

- L'information type_paire est une constante égale ici à « table », indiquant que la paire contient les informations relatives à une table de données ;
- L'information nom_table est le nom « métier » attribué à la table de données dans laquelle est stockée la paire.
  Trois tables coexistent dans le présent exemple : les « routes courantes », les « routes par défaut » et les « ordres de déroutage » ;
- L'information clé_primaire est la clé « métier » utilisée pour identifier la donnée de la paire de façon unique dans la table nom_table.
  Dans le présent exemple, la clé primaire correspond à l'adresse IP atm_ip de l'automate (110) pour la table « routes courantes », à l'identifiant atm_id de l'automate (110) pour la table « ordres de déroutage » et à la version atm_version du logiciel pour la table « routes par défaut ».

Le stockage des données concerne entre autres les routes par défaut entre les automates (110) et les serveurs de traitement (120), les routes courantes des automates (110) ayant une session ouverte avec un serveur de traitement (120) et les éventuels ordres de déroutage pour les automates ayant une session ouverte avec un serveur de traitement (120).

Il convient de noter que la table des routes courantes peut comprendre des routes des automates (110) ayant précédemment eu une session ouverte avec un serveur de traitement (120).

Les bases de données (136) peuvent également stocker des informations secondaires comme par exemple des statistiques sur les performances de fonctionnement, la qualité de service ou les erreurs.

Les routes par défaut sont déterminées pour chaque version atm_version du logiciel des automates (110). Les routes par défaut indiquent les serveurs de traitement (120) pouvant accepter des requêtes provenant d'un automate (110) ayant une version atm_version donnée. Il convient de noter que seuls les serveurs de traitement disponibles, c.-à-d. qui ne sont pas incidentés ou qui n'ont pas été suspendus volontairement, sont indiqués par les routes par défaut.

Chaque route courante est associée à l'adresse IP atm_ip d'un automate (110), et comprend la route actuelle de l'automate (110), l'identifiant atm_id de l'automate (110), la version atm_version de l'automate (110) et l'horodatage de la dernière requête provenant de l'automate (110).

Les éventuels ordres de déroutage sont associés chacun à l'identifiant atm_id d'un automate (110). Les ordres de déroutage indiquent la nouvelle route à suivre pour une requête provenant de l'automate (110) correspondant, ainsi qu'un indicateur d'urgence de l'ordre de déroutage qui dépend du motif de déroutage. Par exemple, si un serveur de traitement (120) est incidenté alors qu'une session est en cours avec un automate (110), l'ordre de déroutage d'une requête provenant de cet automate est immédiat afin de rétablir au plus vite une nouvelle session avec un nouveau serveur de traitement (120). A l'inverse, si l'ordre de déroutage n'est pas immédiat, le déroutage n'intervient que lorsque la session de l'utilisateur de l'automate (110) est détectée terminée, ou lorsque cette session dépasse un délai maximum (« timeout » en anglais) configurable.

Il convient de noter que l'ordre de déroutage peut indiquer que les requêtes provenant d'un automate (110) sont refusées. Par exemple, les requêtes sont refusées lorsqu'un automate (110) est défectueux ou lorsqu'aucun serveur de traitement (120), compatible avec la version du logiciel de l'automate (110), n'est disponible pour accepter la requête de l'automate (110). Les requêtes peuvent également être refusées à la suite d'une décision d'un opérateur supervisant l'échange

des données entre les automates (110) et les serveurs de traitement (120), ou pour éviter une propagation en cascade de déroutages en masse provoqués par des incidents graves ou multiples.

A partir de l'adresse IP de l'automate (111a) à l'origine de la requête, l'application informatique de routage fait appel aux fonctions d'une bibliothèque logicielle de stockage du RAM-Cloud (134) afin de lire ou d'écrire les informations liées au routage. La bibliothèque logicielle, également connue sous le nom de librairie informatique, comprend une fonction de lecture consolidée des données et une fonction d'écriture consolidée des données. L'application informatique de routage peut ainsi demander quelle est la route courante associée à l'adresse IP atm_ip de l'automate (111a), ou si un ordre de déroutage associé à l'identifiant atm_id de l'automate (111a) existe. Dans le cas où aucune route courante n'est associée à l'automate (111a), l'application informatique de routage peut choisir une route par défaut parmi les routes par défaut associées à la version atm_version de l'automate (111a).

Dans le présent exemple, les routes courantes sont stockées dans la table « HR », les routes par défaut dans la table « DR » et les ordres de déroutage dans la table « NR ».

Les données stockées dans le RAM-Cloud (134) sont avantageusement dupliquées sous la forme de trois (R) réplicas dans trois bases de données (136a) par l'intermédiaire de la fonction d'écriture consolidée de la bibliothèque logicielle de stockage.

Dans le présent exemple comprenant quatre nœuds informatiques (135), appelés par la suite host_a, host_b, host_c et host_d, une donnée associée à un automate (110) ayant une adresse IP atm_ip est stockée dans une table de la base de données de trois des quatre nœuds (135). Le stockage de la donnée dans les trois bases de données (136a), formant la liste des réplicas, s'effectue sous la forme d'une paire notée { clé k ; valeur v = (objet x, horodatage h) }, l'objet x contenant la donnée associée à l'automate, l'horodatage h étant inséré par la fonction d'écriture consolidée, la clé composite k comprenant d'une part la clé primaire associée à la donnée pour la table dans laquelle est stockée la donnée et d'autre part le type et le nom de la table dans laquelle est stockée la donnée.

Il convient de noter que la clé primaire dépend de la table et correspond dans le présent exemple à l'adresse IP atm_ip de l'automate pour la table « HR » des routes courantes, à l'identifiant atm_id de l'automate pour la table « NR » des ordres de déroutage, et à la version atm_version de l'automate pour la table « DR » des routes par défaut.

*****

La Figure Formelle 2 représente une variante de ce mode de réalisation particulier de l'invention où les nœuds informatiques (135) du système informatique en nuage de type RAM-Cloud (134) correspondent chacun à un serveur de routage (133).

Ainsi chaque nœud informatique traite l'application informatique de routage et un moteur de base de données. En d'autres termes, les quatre nœuds informatiques de routage et les quatre nœuds des données sont mutualisés. Cette variante correspond à un mode de réalisation optimal de l'invention.

*****

La Figure Formelle 3 représente sous la forme d'un schéma synoptique un procédé d'écriture consolidée (200) d'une paire dans le RAM-Cloud (134). Le procédé d'écriture consolidée (200), utilisé par la fonction de routage c.-à-d. par la fonction cliente du nuage, commence par déterminer, lors d'une première étape (210) la première base de données (136a) de la liste des réplicas sur lesquels est stockée la paire. La première base (136a) est déterminée par l'intermédiaire d'une fonction de hachage sur la clé de la paire. La fonction de hachage permet de distribuer les paires de manière uniforme entre les quatre bases (136) du RAM-Cloud (134).

Dans le présent exemple, la fonction de hachage, uniforme et consistante pour un RAM-Cloud (134) comportant N nœuds, vaut :

$$Hash(k) = \left\lfloor \frac{MD5(k)\ modulo\ 10^4}{10^4} * N \right\rfloor\ ;\ Hash(k)\ \in\ [0, N[\ ;\ N \ll 10^4$$

La fonction MD5 (« Message Digest 5 » en anglais) est une fonction de hachage cryptographique connue de l'homme de métier.

Il convient de noter que la fonction de hachage est également consistante. En effet, en cas d'une modification du nombre de nœuds informatiques (135) du système informatique en nuage (134), la fonction de hachage s'adapte automatiquement à cette nouvelle configuration. Une partie des paires préalablement stockées est également redistribuée entre les différents nœuds informatiques (135). Le nombre de paires subissant une redistribution est au maximum égal au nombre total de clés sur le nombre de nœuds informatiques (135).

La deuxième et la troisième base (136a) de la liste des réplicas sont déterminées, lors d'une deuxième étape (220) du procédé d'écriture, comme étant hébergées sur les nœuds suivants celui de la première base dans la liste { host_a, host_b, host_c, host_d }. Par exemple, si la fonction de hachage détermine que le premier réplica est écrit dans la base sur host_c, alors le deuxième réplica correspond à host_d et le troisième réplica à host_a. Il convient de noter que la liste est dite cyclique c.-à-d. qu'elle recommence automatiquement au début lorsque la fin de la liste est atteinte.

Par ailleurs, il convient également de noter qu'étant donné que chaque nœud informatique (135) comprend une seule base (136), la liste { host_a, host_b, host_c, host_d } correspond à la fois à la liste des nœuds et à la liste des bases. Dans le cas plus général où les nœuds comprennent plusieurs bases, la liste des réplicas est déterminée parmi la liste des bases.

Le procédé d'écriture consolidée insère dans une troisième étape (230) un horodatage en millisecondes ou en microsecondes dans les données de la paire à stocker dans le RAM-Cloud (134). Il convient de noter que l'horodatage est écrit à l'identique dans chaque réplica des trois bases (136a).

L'écriture des réplicas de la paire à stocker s'effectue de manière synchrone sur les bases (136a) de la liste des réplicas, lors d'une quatrième étape (240) du procédé d'écriture consolidée (200).

L'écriture consolidée est considérée comme effectuée dès qu'un quorum d'écritures de réplicas est atteint. Dans le présent exemple, le quorum est atteint lorsque l'écriture est réussie sur deux des trois bases (136a), correspondant à plus de la moitié des réplicas ou à une majorité absolue des réplicas.

Il convient de noter que la fonction d'écriture consolidée de la bibliothèque logicielle de stockage est opérée sur un des nœuds informatiques (135) du RAM-Cloud (134), appelé par la suite nœud local d'écriture. Ainsi, si le nœud local d'écriture est compris dans la liste des réplicas, l'écriture commence alors par le nœud local d'écriture. Ainsi, par exemple, si la fonction d'écriture est traitée sur le nœud host_d sur lequel est stocké le deuxième réplica, l'écriture démarre par le nœud local host_d afin d'obtenir une première écriture réussie dans un très bref laps de temps.

Cet accès prioritaire au nœud local est particulièrement utilisé dans le mode de réalisation optimal illustré en Figure Formelle 2, où les nœuds informatiques (135) du RAM-Cloud (134) sont mutualisés avec les serveurs de routage (133). En d'autres termes, dans ce mode de réalisation optimal, une instance de l'application informatique de routage appelant la fonction d'écriture consolidée est traitée localement sur un des nœuds informatiques (135) du RAM-Cloud (134).

Dès que le quorum est atteint, l'écriture des réplicas restants est effectuée en asynchrone lors d'une cinquième étape (250) du procédé d'écriture consolidée (200).

Il convient de noter que le temps de réponse indiquant que l'écriture des réplicas est réussie correspond aux écritures réussies en synchrone sur le quorum des bases (136a). En d'autres termes, le procédé d'écriture consolidée (200) permet de minimiser le temps d'écriture vu par l'application informatique de routage ayant commandée l'écriture consolidée d'une paire dans le RAM-Cloud (134).

En outre, le procédé d'écriture consolidée (200) contribue à la forte résilience du RAM-Cloud (134), quels que soient le volume des données à traiter et le nombre de nœuds. En effet, lorsqu'un nœud est en défaut suite par exemple à une panne technique, ce nœud est automatiquement mis de côté et l'écriture d'une paire sur ce nœud peut être effectuée en asynchrone dès lors que le nœud est de nouveau disponible.

Dans des variantes de ce mode de réalisation particulier, le procédé d'écriture consolidée des données comprend une étape d'écriture d'un historique des modifications des données. Les modifications des données sur un nœud sont écrites séquentiellement dans un fichier présent sur le nœud correspondant. Cet historique des modifications des données permet de régénérer les données du stockage dans la mémoire du nœud, ce qui peut être utile par exemple dans le cas où le nœud est indisponible pendant un laps de temps.

L'application informatique de routage fait également appel à la fonction de lecture consolidée de la bibliothèque logicielle de stockage pour lire des informations de routage liées à l'identifiant du distributeur automatique (111a).

*****

La Figure Formelle 4 et la Figure Formelle 5 sont deux moitiés d'un même schéma synoptique qui représente un procédé hystérétique probabiliste rétroactif intégré de lecture consolidée (300) d'une paire dans le RAM-Cloud (134). Le présent exemple ainsi schématisé comprend plusieurs éléments qui inter-opèrent :

- Une instance (301) de l'application de routage invoquant la fonction de lecture consolidée LC4 d'une paire (340), ladite fonction étant comprise dans le procédé intégré LC4 (300), ledit procédé étant compris dans la bibliothèque logicielle de stockage ;
- Le procédé intégré LC4 (300) ou procédé hystérétique probabiliste rétroactif intégré de lecture consolidée d'une paire dans le nuage de type RAM-Cloud, ledit procédé comprenant plusieurs constituants en interaction et/ou rétroaction, à savoir :
    - La paramétrisation LC4 (302) ;
    - La fonction ou procédé de lecture consolidée LC4 (340) comprenant :
        - Le prologue LC4 (340a) ;
        - Le corps LC4 (340b) ;
    - Le procédé hystérétique (310) ;
    - La table TableEtats (320) ;
    - La fonction de probabilité LoiTransition (330).

Le présent exemple correspond au mode de réalisation optimal dans lequel les nœuds de routage (133) opérant l'application de routage c.-à-d. l'application cliente du nuage, sont mutualisés avec les nœuds (135) du nuage (134). Ainsi, chaque nœud opère une ou des instances de l'application de routage plus un moteur et une base de données.

Le nuage (134) comprend quatre (N) nœuds mutualisés dont la liste déterminée par configuration est { host_a, host_b, host_c, host_d }.

Une instance (301) de l'application cliente du nuage, opérée sur un nœud invoquant c.-à-d. sur host_a (133a) dans l'exemple, invoque la lecture consolidée d'une paire et opère le procédé intégré LC4 (300) de la bibliothèque logicielle de stockage.

Seuls sont schématisés les trois nœuds { host_a, host_c, host_d } hébergeant les trois bases (136a) stockant les trois (R) réplicas de la paire dont la lecture consolidée est invoquée par

l'instance (301). L'un des réplicas est stocké dans la base (136a) hébergée sur le serveur invoquant host_a (133a).

Toutes les instances de l'application de routage opérées sur tous les nœuds { host_a, host_b, host_c, host_d } peuvent invoquer les fonctions de lecture et d'écriture consolidées des paires, se traduisant par des lectures (359) et des écritures de réplicas sur tous les nœuds { host_a, host_b, host_c, host_d } constituant le nuage (134).

Les écritures de réplicas alimentent de façon négligeable le procédé hystérétique (310), alors que les lectures (359) de réplicas effectuées (355) par la lecture consolidée LC4 (340) alimentent (311) de façon majoritaire le procédé hystérétique (310).

Pour chaque nœud invoquant de la liste { host_a, host_b, host_c, host_d }, le procédé hystérétique opère trois variables (c.s, e.s, i.s) sur le nœud invoquant, pour chaque nœud s ayant déjà fait l'objet d'au moins une lecture ou d'une écriture d'un réplica par le nœud invoquant.

En particulier dans l'exemple, le procédé hystérétique (310) opère sur le nœud invoquant host_a (133a) :

- Les variables (c.host_a, e.host_a, i.host_a) correspondant respectivement au compteur d'erreurs, à l'état hystérétique et à l'horodatage de la dernière inversion dudit état, qui sont associées au nœud host_a ;
- Les variables (c.host_b, e.host_b, i.host_b) correspondant respectivement au compteur d'erreurs, à l'état hystérétique et à l'horodatage de la dernière inversion dudit état, qui sont associées au nœud host_b ;
- Les variables (c.host_c, e.host_c, i.host_c) correspondant respectivement au compteur d'erreurs, à l'état hystérétique et à l'horodatage de la dernière inversion dudit état, qui sont associées au nœud host_c ;
- Les variables (c.host_d, e.host_d, i.host_d) correspondant respectivement au compteur d'erreurs, à l'état hystérétique et à l'horodatage de la dernière inversion dudit état, qui sont associées au nœud host_d.

Lorsque l'instance (301) de l'application de routage opérée sur le nœud invoquant host_a (133a), invoque la fonction de lecture consolidée LC4 (340) de la bibliothèque logicielle de stockage, ladite fonction opérée sur le nœud invoquant host_a (133a) effectue des lectures (359) des réplicas stockés sur les bases (136a) hébergées sur les nœuds (135) du nuage (134).

Pour chaque requête en lecture (359) d'un réplica sur un nœud s, invoquée par l'instance (301) opérée sur le nœud invoquant host_a (133a), le procédé hystérétique (310) opéré sur le nœud invoquant host_a (133a) observe le statut { en erreur ; non en erreur } (311) de la requête (359).

Selon que le statut (311) de la requête est en erreur ou respectivement non en erreur, la condition technique de s est en erreur ou respectivement non en erreur du point de vue du nœud invoquant host_a (133a).

Selon que la condition technique de s est en erreur ou respectivement non en erreur du point de vue du nœud invoquant host_a (133a), le compteur d'erreurs c.s associé à s et opéré sur le nœud invoquant host_a (133a) est incrémenté ou respectivement décrémenté d'une unité (310). La valeur dudit compteur est bornée sur l'intervalle [0, E].

Ce compteur d'erreurs c.s associé à s et opéré sur le nœud invoquant host_a (133a) permet de déterminer d'une part l'état hystérétique e.s associé à s et opéré sur le nœud invoquant host_a (133a) et d'autre part l'horodatage i.s associé à s et opéré sur le nœud invoquant host_a (133a) correspondant à la dernière inversion de l'état hystérétique, grâce au procédé suivant (310) :

- Lorsque le compteur c.s est incrémenté de la valeur E-1 à E et si l'état e.s est égal à « disponible » (310a), alors e.s reçoit la valeur « indisponible » et i.s reçoit l'horodatage de maintenant c.-à-d. de la date et de l'heure courantes ;
- Lorsque le compteur c.s est décrémenté de la valeur 1 à 0 et si l'état e.s est égal à « indisponible » (310b), alors e.s reçoit la valeur « disponible » et i.s reçoit l'horodatage de maintenant c.-à-d. de la date et de l'heure courantes ;
- Dans les autres cas, les variables e.s et i.s demeurent inchangées.

Chaque fois que l'état e.s change c.-à-d. qu'il s'inverse (310a, 310b), le triplet (s, e.s, i.s) est écrit (312) dans la table TableEtats (320) opérée sur le nœud invoquant host_a (133a). Ladite table (320) contient au plus un triplet (s, e.s, i.s) pour chaque valeur de s possible. L'opération d'écriture (312) consiste soit à insérer dans la table le triplet (s, e.s, i.s) si la table ne contient pas déjà un triplet associé à s, soit dans le cas contraire à mettre à jour dans la table les informations e.s et i.s dans le triplet existant associé à s.

Dans l'exemple, la table TableEtats (320) opérée sur le nœud invoquant host_a (133a) comprend un triplet associé à chaque nœud de la liste des nœuds { host_a, host_b, host_c, host_d } c.-à-d. quatre triplets (host_a, e.host_a, i.host_a), (host_b, e.host_b, i.host_b), (host_c, e.host_c, i.host_c), (host_d, e.host_d, i.host_d) correspondant par exemple à la table TableEtats suivante :

Nœud	**État hystérétique du nœud**	**Horodatage de la dernière inversion d'état du nœud**
host_a	disponible	1487533881.826 (19/02/2017 20h 51mn 21s 826ms)
host_b	indisponible	1487933247.047 (24/02/2017 11h 47mn 27s 47ms)
host_c	disponible	1487631739.247 (21/02/2017 00h 02mn 19s 247ms)
host_d	disponible	1487933954.102 (24/02/2017 11h 59mn 14s 102ms)

Ce procédé hystérétique (310) opéré sur le nœud invoquant host_a (133a), alimente (312) la table TableEtats (320) opérée sur le nœud invoquant host_a (133a), laquelle table alimente (321) la lecture consolidée LC4 (340).

Dans l'exemple, l'instance (301) de l'application de routage souhaitant lire dans le nuage (134) une paire dont la clé est k, invoque le prologue LC4 (340a) en lui transmettant la clé k plus éventuellement d'autres paramètres au cours de l'étape de paramétrisation LC4 (302).

L'instance (301) de l'application de routage connait les particularités, la sensibilité et le contexte des différentes tables de données, voire même d'une ou de plusieurs paires en particulier. L'application de routage peut, pour tout ou partie des paires, à tout moment, modifier son exigence en termes de fiabilité ou de performance par l'intermédiaire du paramétrage transmis.

Ainsi, en transmettant un paramètre optionnel full_scan ou f, l'instance (301) de l'application de routage peut influencer la lecture consolidée LC4 (340) afin de maximiser la fiabilité de la lecture consolidée d'une paire, en forçant la lecture de tous les réplicas afin que celui dont l'horodatage est le plus récent soit sélectionné.

A l'opposé, en transmettant un paramètre optionnel joker ou j, l'instance (301) de l'application de routage peut influencer la lecture consolidée LC4 (340) afin de maximiser la performance de la lecture consolidée d'une paire sans pour autant perdre en fiabilité. Dans ce

mode, un réplica dont l'horodatage est plus récent qu'une limite spécifiée par l'intermédiaire du paramètre j, est réputé fiable et dispense de lire davantage de réplicas.

Le joker est déterminé pour permettre d'accepter automatiquement une paire ayant un jeune âge c.-à-d. ayant été créée ou mise à jour récemment. Le joker est déterminé en fonction de la fréquence avec laquelle les données sont écrites/rafraichies dans la base de données. Par exemple, dans le cas où une route par défaut est rafraîchie chaque minute, alors un réplica dont l'âge est inférieur à une minute est très probablement à jour. Dans ce cas, la valeur du joker peut être déterminée à soixante secondes. L'instance (301) de l'application de routage peut adapter la valeur du joker à chaque requête de lecture consolidée d'une paire. La valeur du joker peut donc changer à chaque requête pour chaque paire donc chaque clé et à fortiori être différente pour chaque table. Une valeur du joker s'exprime généralement en secondes, plus rarement en minutes.

La paramétrisation LC4 (302) se terminant, est suivie du prologue LC4 (340a).

Le prologue LC4 (340a) débute par une étape (341) d'initialisation inconditionnelle des variables (r+, r-, x', h', q, m), comprenant les actions suivantes :

- La variable r+ comptant les réponses positives des lectures de réplicas est initialisée à zéro ;
- La variable r- comptant les réponses négatives des lectures de réplicas est initialisée à zéro ;
- La variable x' correspondant à l'objet de référence est initialisée à néant ;
- La variable h' correspondant à l'horodatage de référence, c.-à-d. à l'horodatage de l'objet de référence, est initialisée à zéro ;
- La variable q contenant la valeur du quorum applicable pendant la lecture consolidée de la paire, est initialisée égale à Q c.-à-d. à deux dans l'exemple ;
- La variable m est initialisée à l'horodatage de maintenant c.-à-d. de la date et de l'heure courantes. Toutes les étapes du corps LC4 (340b) qui nécessitent de connaitre l'horodatage de maintenant se réfèrent à cette variable. Ainsi le moment présent est invariant pendant l'exécution d'une lecture consolidée.
  Par exemple, la variable m peut recevoir la valeur 1487933965.844, la partie en préfixe du point étant au format dit « EPOCH » c.-à-d. correspondant au nombre de secondes écoulées depuis le 01/01/1970 à minuit, et la partie en suffixe correspondant au nombre de millisecondes écoulées depuis le début de la seconde. Dans l'exemple, la valeur 1487933965.844 correspond au 24/02/2017 à 11h 59mn 25s et 844ms.

Puis le prologue (340a) se poursuit par une seconde étape (342) d'initialisation conditionnelle du paramètre j et de la variable q, comprenant les actions suivantes :

Si le paramètre f est égal à VRAI, alors :

- Le paramètre j, qu'il ait été ou non transmis via l'étape de paramétrisation LC4 (302), est initialisé à zéro ce qui privera d'effet le procédé du joker utilisé par le corps LC4 (340b). Les paramètres full_scan et joker s'excluent mutuellement.
  Lorsque par exemple l'instance (301) invoquant la lecture consolidée d'une paire transmet (302) au prologue (340a) le paramètre f égal à VRAI et le paramètre j égal à une valeur positive non nulle, alors le paramètre j est initialisé à zéro afin de désactiver le procédé du joker car le procédé full_scan est prioritaire.
- La variable q est initialisée à l'infini, ce qui privera le corps LC4 (340b) de la possibilité d'atteindre un quorum et forcera ainsi la lecture de tous les réplicas.

Dans le cas contraire (f est égal à FAUX), alors le paramètre j et la variable q demeurent inchangés.

Puis le prologue LC4 (340a) détermine, à partir de la clé k de la paire, la liste des bases hébergeant les réplicas de la paire :

- La première base de données (136a) de la liste des réplicas est indiquée grâce à la même fonction de hachage que celle utilisée dans le procédé d'écriture consolidée.
  Par exemple, la fonction de hachage appliquée sur la clé k produit la valeur trois, indiquant que le premier réplica est hébergé sur la base qui se trouve en troisième position dans la liste configurée de tous les nœuds { host_a, host_b, host_c, host_d } c.-à-d. qu'il est hébergé sur host_c ;
- La deuxième puis la troisième base (136a) sont déterminées comme étant les éléments suivants de la liste configurée de tous les nœuds, le successeur du dernier élément de la liste dite cyclique étant le premier élément de la liste.
  Dans l'exemple, la deuxième et la troisième base sont respectivement host_d et host_a.

Ainsi, dans l'exemple, la liste des premier, second et troisième réplicas est égale à { host_c, host_d, host_a } et comprend le nœud invoquant host_a (133a).

Enfin, le prologue (340a) initialise (343) une variable ϕ correspondant à une file, avec les noms des nœuds des réplicas de la paire. La file est ordonnée de manière aléatoire, excepté en ce qui concerne le nœud host_a qui est placé en tête de file puisque il correspond dans l'exemple simultanément à un réplica et au nœud invoquant (133a).

Ainsi, la file de l'exemple peut contenir { host_a, host_d, host_c } déterminant la liste des réplicas de la paire et l'ordre prévisionnel dans lequel ils seront lus par le corps LC4 (340b).

Ce positionnement de host_a en tête de la file permettra à host_a d'être lu en premier par le corps LC4 (340b). En d'autres termes ce procédé permettra de lire le premier réplica localement sur host_a, ce qui est doublement avantageux :

- Lorsque le procédé du joker est utilisé de façon nominale, la lecture consolidée peut être statuée en lisant un seul réplica : le fait qu'il soit local accroit encore la performance ;
- Lorsque le procédé du quorum est utilisé, la lecture de deux (Q) réplicas sur trois (R) dans l'exemple permet de statuer la lecture consolidée : lire localement un premier réplica sur host_a permet d'atteindre plus rapidement la lecture d'un second réplica sur host_d, ce qui améliore la performance.

Le prologue LC4 (340a) se terminant, est suivi du corps LC4 (340b).

Le corps LC4 (340b) procède à des itérations, lisant successivement tout ou partie des trois (R) réplicas dans les bases de données indiquées par le contenu de la file ϕ.

La prochaine base de données de la liste des réplicas à lire est déterminée lors de l'étape (351) qui est répétée (356) autant de fois que nécessaire jusqu'à épuisement de la file ϕ (350). Lors de cette étape (351), le choix de la prochaine base (136a) à lire est effectué en sélectionnant la base, ou plus exactement le nœud associé, qui se trouve en tête de la file, puis en supprimant cet élément de la file, laquelle se réduit donc à chaque itération (356). Dans l'exemple, la première itération sélectionne le nœud host_a, premier élément dans la file. Les itérations suivantes sélectionneront host_d puis finalement host_c. Le nœud sélectionné par l'itération en cours, correspondant à la base de données (136a) du réplica à lire, est également appelé nœud courant ou nœud n.

Lors d'une étape suivante (352), le corps LC4 (340b) opère localement sur le nœud invoquant host_a (133a) les actions suivantes :

1. Écrire (352) dans une variable e.n l'état hystérétique du nœud n du point de vue du nœud invoquant host_a (133a). Cette information est préalablement lue (321) dans la table TableEtats (320) opérée localement sur le nœud invoquant host_a (133a) ;
   Par exemple, lors de la seconde itération sur la file ϕ c.-à-d. lorsque le nœud courant n est.host_d, la variable e.host_d opérée localement sur le nœud invoquant host_a (133a) reçoit l'état hystérétique du nœud host_d, ledit état étant au préalable lu (321) dans la table TableEtat (320) opérée localement sur le nœud invoquant host_a (133a). Ainsi, e.host_d peut par exemple recevoir la valeur « disponible » ;
2. Écrire (352) dans une variable i.n l'horodatage de la dernière inversion d'état du nœud n du point de vue du nœud invoquant host_a. Cette information est préalablement lue (321) dans la table TableEtats (320) opérée localement sur le nœud invoquant host_a (133a) ;
   Par exemple, lors de la seconde itération sur la file ϕ c.-à-d. lorsque le nœud courant n est host_d, la variable i.host_d opérée localement sur le nœud invoquant host_a (133a) reçoit l'horodatage de la dernière inversion d'état hystérétique du nœud host_d, ledit horodatage étant au préalable lu (321) dans la table TableEtat (320) opérée localement sur le nœud invoquant host_a (133a). Ainsi, i.host_d peut par exemple recevoir la valeur 1487933954.102 (24/02/2017 11h 59mn 14s 102ms) ;
3. Écrire (352) dans une variable d.n le délai écoulé depuis la dernière inversion d'état hystérétique du nœud n, calculé comme m moins i.n ;
   Par exemple, lors de la seconde itération sur la file ϕ c.-à-d. lorsque le nœud courant n est host_d, la variable d.host_d opérée localement sur le nœud invoquant host_a (133a) reçoit la valeur m moins i.host_d, c.-à-d. par exemple 1487933965.844 (24/02/2017 11h 59mn 25s 844ms) moins 1487933954.102 (24/02/2017 11h 59mn 14s 102ms) ce qui correspond encore à 11.742s. Ainsi, d.host_d peut par exemple recevoir la valeur 11.742 dans l'exemple, indiquant que le nœud host_d a changé d'état hystérétique depuis 11.742 secondes du point de vue du nœud invoquant host_a (133a).

Lors de deux étapes suivantes (353, 354), le corps LC4 (340b) effectue un choix : soit il décide (354) de lire (355) le réplica sur le nœud courant n puis de poursuivre l'itération courante, soit il décide (354) d'ignorer (356) le réplica sur le nœud courant n. Ce choix s'effectue avec une probabilité déterminée par la fonction de probabilité LoiTransition multiparamétrique à partir des trois paramètres qui lui sont transmis, à savoir, l'état e.n hystérétique du nœud n, le délai d.n écoulé depuis la dernière inversion de l'état hystérétique du nœud n, la constante configurable A correspondant à la durée d'amortissement lors des transitions de l'état hystérétique.

Dans l'exemple présent, la fonction de probabilité LoiTransition, qui peut être notée

LoiTransition : (état e, durée d, constante A) → probabilité p

est ainsi définie (330) :

- Si l'état e hystérétique est « indisponible » et si d $\in$ [0, A] alors :
$$p = 1 - (1 - 0.01) * \frac{d}{A} = 1 - \frac{0.99 * d}{A}$$
- Si l'état e hystérétique est « indisponible » et si d $\in$]A, +∞[alors :
$$p = 0.01\ (1\%)$$
- Si l'état e hystérétique est « disponible » et si d $\in$ [0, A] alors :
$$p = 0.01 + (1 - 0.01) * \frac{d}{A} = 0.01 + \frac{0.99 * d}{A}$$
- Si l'état e hystérétique est « disponible » et si d $\in$]A, +∞[alors :
$$p = 1\ (100\%)$$

Le processus de décision se déroule ainsi :

1. Écrire (353) dans une variable p.n la probabilité de lire le réplica sur le nœud n, calculée (331) comme le résultat de la fonction LoiTransition (330) à partir des trois paramètres (e.n, d.n, A) qui lui sont transmis.
   Si par exemple l'état hystérétique e.n est égal à « indisponible », et d.n est égal à 20 secondes, et A est égal à 30 secondes, alors :
   $$p.n = LoiTransition(e.n, d.n, A) = 1 - \frac{0.99 * d.n}{A} \cong 0.34$$
   La probabilité de lire le réplica du nœud courant n dans l'exemple est de 34%.
2. Effectuer (353) un tirage aléatoire d'un nombre réel z sur l'intervalle [0, 1] ;
3. Si ($z \leq p.n$) (354) alors poursuivre à l'étape de lecture du réplica (355) ci-après.
   Dans le cas contraire ($z > p.n$) alors ignorer le nœud n puis itérer (356) le corps LC4 (340b) sur le prochain nœud dans la file ϕ.
   Si par exemple p.n est égal à 0.34, alors il existe une probabilité de 34% pour que le réplica soit lu sur le nœud n, et de 66% pour qu'il ne soit pas lu.

Les propriétés de la fonction LoiTransition (330), bornée sur l'intervalle [1/100, 1] avec des transitions inter-états linéaires croissantes ou décroissantes de durée A, sont avantageuses.

En effet, d'une part, la fonction détermine une probabilité non nulle et en particulier supérieure à un centième c.-à-d. à 1%. Puisque au moins 1% des requêtes en lecture sont effectuées (359), le procédé hystérétique (310) est toujours alimenté (311), ce qui dispense de mettre en place un système de supervision.

De plus, le transfert progressif décroissant de la charge a pour effet que le temps de réactivité de l'état hystérétique reste toujours négligeable relativement à la durée de l'évènement déclencheur. A titre d'exemples :

- Suite à un incident récent affectant le nœud n, l'état e.n hystérétique de n est « indisponible » depuis un délai d.n = 3s inférieur à A. La probabilité p.n de lire le réplica sur le nœud n est alors égale à 90%. Si 10000 requêtes/s sont invoquées vers le nœud n, alors 9000 requêtes/s sont encore transmises vers n. Si l'incident termine maintenant, le délai nécessaire au procédé hystérétique pour décider que l'état e.n redevient « disponible » est égal à E/9000 c.-à-d. 0.000333s.
  Le délai de décision d'un retour en service après cette courte indisponibilité, est négligeable au regard de la durée de l'indisponibilité (0.000333 << 3).
- Suite à un incident ancien affectant le nœud n, l'état e.n hystérétique de n est « indisponible » depuis un délai d.n = 3600s supérieur à A. La probabilité p.n de lire le réplica sur le nœud n est alors égale à 1%. Si 10000 requêtes/s sont invoquées vers le nœud n, alors 100 requêtes/s sont encore transmises vers n. Si l'incident termine maintenant, le délai nécessaire au procédé hystérétique pour décider que l'état e.n redevient « disponible » est égal à E/100 c.-à-d. 0.03s.
  Le délai de décision d'un retour en service après cette longue indisponibilité, est négligeable au regard de la durée de l'indisponibilité (0.03 << 3600).

Enfin, le transfert progressif croissant de la charge réduit le risque d'incident inhérent à une remise en charge brutale d'une base de données.

Lors d'une étape suivante (355), le corps LC4 lit (359) le réplica associé à la clé k sur le nœud n. Le réplica peut être noté { clé k ; valeur v.n = (objet x.n, horodatage h.n) }, v.n étant la valeur lue associée à la clé, v.n contenant l'objet x.n et l'horodatage h.n, x.n étant l'objet fournit par l'application de routage lors de l'écriture consolidée de la paire, h.n étant l'horodatage inséré à l'identique dans les réplicas de la paire par la fonction d'écriture consolidée.

L’instance (301) de l’application de routage est opérée sur host_a (133a). Le fonctionnement du procédé hystérétique (310) alimenté (311) par la lecture (359) peut être illustré par ces exemples :

- Si par exemple le statut (311) de la lecture (359) d’un réplica sur host_d est en erreur, et si le compteur c.host_d opéré sur host_a est égal à zéro, alors c.host_d est incrémenté d’une unité donc devient égal à un, sans autre effet ;
- Si par exemple le statut de la lecture (359) d’un réplica sur host_d est en erreur, et si le compteur c.host_d opéré sur host_a est égal à deux (c.-à-d. à E-1), alors c.host_d est incrémenté d’une unité donc devient égal à E. En conséquence, mais seulement si l’état e.host_d opéré sur host_a est « disponible » (310a), alors l’état e.host_d opéré sur host_a devient « indisponible » et l’horodatage i.host_d de la dernière inversion d’état venant de se produire devient égal à l’horodatage de maintenant c.-à-d. de la date et de l’heure courantes ;
- Si par exemple le statut (311) de la lecture (359) d’un réplica sur host_d est en erreur, et si le compteur c.host_d opéré sur host_a est égal à E, alors le procédé hystérétique n’a pas d’effet ;
- Si par exemple le statut (311) de la lecture (359) d’un réplica sur host_d n’est pas en erreur, et si le compteur c.host_d opéré sur host_a est égal à deux, alors c.host_d est décrémenté d’une unité donc devient égal à un, sans autre effet ;
- Si par exemple le statut (311) de la lecture (359) d’un réplica sur host_d n’est pas en erreur, et si le compteur c.host_d opéré sur host_a est égal à un, alors c.host_d est décrémenté d’une unité donc devient égal à zéro. En conséquence, mais seulement si l’état e.host_d opéré sur host_a est « indisponible » (310b), alors l’état e.host_d opéré sur host_a devient « disponible » et l’horodatage i.host_d de la dernière inversion d’état venant de se produire devient égal à l’horodatage de maintenant c.-à-d. de la date et de l’heure courantes ;
- Si par exemple le statut (311) de la lecture (359) d’un réplica sur host_d n’est pas en erreur, et si le compteur c.host_d opéré sur host_a est égal à zéro, alors le procédé hystérétique n’a pas d’effet.

Lorsque l’étape (355) de lecture du réplica excède en durée la valeur de la constante configurable T positive non nulle, alors le corps LC4 (340b) interrompt immédiatement l’étape de lecture (355) puis abandonne l’itération en cours et itère (356) sur le prochain nœud dans la file ϕ. Ce procédé garantit la durée de lecture d'un réplica, ce qui à son tour garantit la durée du procédé de lecture consolidée d’une paire, ce qui permet de traiter en temps réel les requêtes en lecture provenant de l’application de routage.

Lorsque l’étape (355) de lecture du réplica provoque une erreur, alors le corps LC4 (340b) abandonne l’itération en cours et itère (356) sur le prochain nœud dans la file ϕ. L’erreur de lecture d’un réplica peut traduire l’indisponibilité d’un moteur de base de données, l’indisponibilité du nœud informatique (135) hébergeant le moteur de base de données, un problème réseau, voire une donnée corrompue.

Lorsque l’étape (355) de lecture du réplica est négative (358), alors :

1. Si (360) l’état e.n hystérétique du nœud est « disponible » et si (d.n < D) alors le nœud n est en rémanence donc la présente lecture négative sur n est considérée comme un faux négatif. En conséquence, ignorer le nœud n puis itérer (356) le corps LC4 (340b) sur le prochain nœud dans la file ϕ ;
   Dans le cas contraire (le nœud n n’est pas en rémanence) continuer au point 2 ci-après (361) ;

2. Incrémenter (361) d'une unité la variable r- comptant les réponses négatives ;
3. Si (r- < q) alors le quorum des réponses négatives n'est pas atteint. En conséquence, itérer (356) le corps LC4 (340b) sur le prochain nœud dans la file ϕ ;
   Dans le cas contraire (r- $\geq$ q) continuer au point 4 ci-après (363) ;
4. Le quorum q des réponses négatives est atteint. En conséquence, la lecture consolidée LC4 (340) termine prématurément la requête en statuant une réponse négative (363).

Si par exemple le nœud courant n correspond à host_d et si e.n c.-à-d. e.host_d est « disponible » et si d.n c.-à-d. d.host_d est égal à 11.742s donc inférieur à D, alors la lecture négative du réplica sur host_d est interprétée comme un probable faux négatif qui ne doit pas être pris en compte dans le décompte des réponses négatives. Ce réplica ne pouvant être contributif au résultat de la lecture consolidée, est ignoré.

Lorsque l'étape (355) de lecture du réplica est positive (357), alors :

1. Si (j > 0) (370) alors le procédé du joker est opérant. En conséquence, continuer au point 2 ci-après (371) ;
   Dans le cas contraire (j = 0), continuer au point 3 ci-après (380) ;
2. Le procédé du joker est opérant et se déroule comme suit :
   a. Déterminer (371) l'âge a.n du réplica lu sur le nœud n, calculé comme m moins h.n.
      Par exemple l'âge a.n du réplica est égal à l'horodatage 1487933965.844 moins l'horodatage 1487933904.145 c.-à-d. 61.699s, correspondant au délai écoulé depuis la création ou la mise à jour de ce réplica ;
   b. Si (a.n < j) (372) alors le réplica lu sur le nœud n est plus jeune que la valeur du joker, il est donc réputé fiable. En conséquence, la lecture consolidée LC4 termine prématurément en statuant une réponse positive. La paire réponse notée { clé k ; valeur = (objet x) } comprend l'objet x lu sur le réplica du nœud n ;
      Dans le cas contraire (a.n $\geq$ j) continuer au point 3 ci-après (380) ;
3. Si (h.n < h') (380) c.-à-d. si l'horodatage du réplica lu sur le nœud n est inférieur à l'horodatage de référence, alors le réplica lu sur le nœud n est obsolète, ce qui constitue un cas d'incohérence des réplicas. En conséquence, ignorer le nœud n puis itérer (356) le corps LC4 (340b) sur le prochain nœud dans la file ϕ ;
   Dans le cas contraire (h.n $\geq$ h') continuer au point 4 ci-après (381) ;
4. Si (h.n > h') (381) c.-à-d. si l'horodatage du réplica lu sur le nœud n est supérieur à l'horodatage de référence, alors le compteur r+ des réponses positives a jusqu'à présent comptabilisé des réplicas obsolètes, ce qui constitue un cas d'incohérence des réplicas. En conséquence continuer au point 5 ci-après (384) ;
   Dans le cas contraire (h.n $\leq$ h') continuer au point 6 ci-après (382) ;
5. Remettre (384) à zéro le compteur r+ ;
6. Incrémenter (382) d'une unité la variable r+ comptant les réponses positives ;
7. Copier (382) l'objet x.n dans l'objet de référence x' et copier (382) l'horodatage h.n dans l'horodatage de référence h' ;
8. Si (r+ < q) (383) alors le quorum des réponses positives n'est pas atteint. En conséquence, itérer (356) le corps LC4 (340b) sur le prochain nœud dans la file ϕ ;
   Dans le cas contraire (r+ $\geq$ q) continuer au point 9 ci-après (385) ;
9. Le quorum des réponses positives est atteint (385). En conséquence, la lecture consolidée LC4 (340) termine prématurément la requête en statuant une réponse

positive. La paire réponse notée { clé k ; valeur = (objet x') } comprend l'objet de référence x' ;

Lorsque le corps LC4 (340b) a épuisé (350) la file ϕ sans être parvenu à statuer prématurément une réponse, alors :

1. Si (r+ > 0) (390) alors il existe au moins une réponse positive. En conséquence, la lecture consolidée LC4 (340) statue une réponse positive. La paire réponse notée { clé k ; valeur = (objet x') } comprend l'objet de référence x' ;
   Dans le cas contraire (r+ = 0) continuer au point 2 ci-après (391) ;
2. Si (r- > 0) (391) alors il existe au moins une réponse négative. En conséquence, la lecture consolidée LC4 (340) statue une réponse négative ;
   Dans le cas contraire (r- = 0) continuer au point 3 ci-après (392) ;
3. Sinon (392), il n'existe aucune réponse positive ni négative c.-à-d. que tous les réplicas lus sont soit en erreur soit en dépassement du délai maximum de lecture. En conséquence, la lecture consolidée LC4 statue une réponse en erreur (392).

La Figure Formelle 4 et la Figure Formelle 5 font apparaitre le fonctionnement rétroactif du procédé intégré LC4 (300). La lecture (359) des réplicas alimente (311) le procédé hystérétique (310) qui alimente (312) à son tour la table TableEtats (320), laquelle alimente (321) d'une part le calcul de la rémanence (360) et d'autre part les paramètres transmis (331) à la fonction de probabilité LoiTransition (330), les deux ayant pour effet de modifier la cinématique des lectures (359) des réplicas lus, et ainsi de suite.

Il convient de noter que le procédé intégré LC4 (300) permet de gérer tous les cas d'incohérences positives ou négatives des données. Le procédé LC4 est en outre résilient et cohérent, tout en présentant des performances maximales.

*****

Afin de permettre un accès direct à l'ensemble des paires stockées dans le nuage (134) sans connaître les clés des paires, l'ensemble des clés des paires peuvent être enregistrées dans une table d'index multiplanaire stockée dans le nuage (134).

Il convient de noter que l'application de routage effectuant la demande d'indexation d'une clé, choisit d'indexer ou non, c'est-à-dire d'enregistrer ou non, une clé dans la table d'index. Ainsi la table d'index peut stocker une partie ou la totalité des clés existantes dans une table de données. Par exemple, l'application de routage peut recevoir une requête émise par un automate (110) considéré comme nouveau car présentant une adresse IP atm_ip inconnue dans la table des routes courantes. Une route courante associée à la clé atm_ip est alors créée, puis une indexation de la clé atm_ip est effectuée, cette indexation n'étant par la suite plus nécessaire pour cette clé. L'indexation d'une clé effectuée à la demande par l'application de routage apporte ainsi performance et souplesse au procédé d'échange de données.

La table d'index est divisée en quatre plans d'index, correspondant chacun à un des quatre serveurs de routage (133). Chaque plan d'index est divisé par exemple en seize (F) fragments d'index. Chaque fragment d'index comprend environ un seizième des clés des paires des données puisque ces clés y sont distribuées par une fonction de hachage uniforme.

Il convient de noter que le nombre de fragments peut être différent sur chaque table d'index et qu'il peut être adapté en fonction du nombre total de clés qui existent dans la table indexée. Ainsi lorsqu'un moteur de base de données KVS présente une limite sur la taille d'une paire stockée, le nombre de clés stockées dans chaque fragment lié à ce moteur KVS est configuré pour satisfaire ce nombre limite.

Chaque fragment d'index comprend des paires d'index

```
{     index_clé  = (type_paire, nom_table, num_fragment, hôte_client) ;
      Index_valeur     = (liste_clés, horodatage)
```

}, où :

- index_clé est une chaîne de caractères comprenant au moins les informations (type_paire, nom_table, num_fragment, hôte_client), où :
    - L'information type_paire est une constante, par exemple « index », indiquant que la paire contient les informations relatives à un index ;
    - L'information nom_table est le nom « métier » attribué à la table de données par l'application de routage, cette table de données étant associée à la clé faisant l'objet de l'indexation. Par exemple, un nom de table peut être « routes courantes » ou « HR » ;
    - L'information num_fragment est le numéro du fragment de l'index ;
    - L'information hôte_client est le nom d'hôte (« hostname » en anglais) du serveur de routage (133) depuis lequel l'application de routage effectue une demande d'indexation d'une clé ;
- index_valeur contient la liste de toutes les clés primaires des paires de la table nom_table qui ont déjà été indexées.

Les clés des paires sont réparties de manière homogène entre les fragments d'index grâce à une fonction de hachage sur la clé.

Par exemple, une instance (301) de l'application de routage, opérée sur le nœud host_a, demande à indexer une clé atm_ip, associée à la table « routes courantes » alias « HR ». La fonction de hachage détermine la valeur num_fragment du fragment correspondant à la clé atm_ip. La clé atm_ip est ensuite indexée c'est-à-dire insérée dans la liste_clés de la paire d'index correspondant à la clé index_clé = (« index », « HR », num_fragment, « host_a »).

Afin d'éviter que deux instances de l'application de routage opérées sur un même serveur host_a de routage (133), écrivent simultanément dans le même fragment d'index du même plan d'index associé à ce serveur de routage (133), un verrou associé à ce plan d'index et à ce fragment est positionné localement sur ledit serveur pour départager les deux instances en concurrence. Ce verrou porte sur le quadruplet (« index », « HR », num_fragment, « host_a ») qui peut être réduit au triplet (« index », « HR », num_fragment) puisque ledit verrou est localisé sur le serveur host_a. Par exemple, un tel verrou s'appuie sur un fichier localisé sur host_a et dont le nom contient la chaîne de caractères « index_HR_2.lock ».

Ainsi lorsque deux instances de l'application informatique opérées sur le même nœud informatique cherchent à écrire en même temps dans le même fragment d'index, le verrou permet de différer une des deux écritures afin qu'elles ne soient pas simultanées.

Par exemple, deux instances de l'application de routage, opérées sur le même nœud host_a, demandent à indexer respectivement une clé atm_ip1 et une clé atm_ip2, associées chacune à la même table « routes courantes » alias « HR ». Dans le cas où la fonction de hachage détermine la même valeur num_fragment du fragment pour les deux clés atm_ip1 et atm_ip2, les clés atm_ip1 et atm_ip2 sont ensuite indexées c'est-à-dire ajoutées séquentiellement dans la même liste_clés de la paire d'index correspondant à la clé d'index index_clé = (« index », « HR », num_fragment, « host_a »). Cette indexation non simultanée est permise par l'utilisation du verrou local sur host_a correspondant à la paire d'index.

Il convient de noter que dans le cas où les deux instances de l'application de routage sont opérées sur des serveurs de routage (133) distincts et cherchent à indexer soit la même clé atm_ip ou soit deux clés associées par la fonction de hachage à la même valeur num_fragment, alors l'indexation n'est pas contrainte par l'utilisation d'un verrou car les deux paires d'index sont distinctes. Chacune de ces paires se trouve en effet dans le plan d'index correspondant au serveur de routage (133) sur lequel est exécuté le processus de routage à l'origine de la demande d'indexation.

Ainsi, les instances opérées sur chaque serveur de routage (133) écrivent dans le plan d'index correspondant au serveur de routage (133) sur lequel elles sont opérées. Ce procédé d'index multiplanaire évite la concurrence entre les serveurs de routage (133) lors des accès en écriture vers les index. Par ailleurs, la lecture de la totalité de l'index, qui consiste à lire toutes les paires d'index de tous les fragments d'index de tous les plans d'index puis à fusionner les listes des clés primaires indexées et à retirer les clés primaires en double, s'effectue également sans concurrence entre les serveurs de routage (133).

Grâce à ce procédé d'index multiplanaire, les opérations d'écritures vers les index (indexation donc ajout d'une clé dans un fragment) comme les opérations de lecture depuis un index (lecture de toutes les clés) s'exécutent sans aucune contention entre les instances de l'application traitées sur des serveurs de routage (133) distincts, ce qui est cohérent avec la meilleure performance et implémente la capacité de mise à l'échelle horizontale.

Il convient de noter que la table d'index peut contenir des clés obsolètes qui sont résolues automatiquement lors de la lecture en vérifiant l'existence des paires auxquelles font référence les clés indexées.

Afin de supprimer régulièrement les clés obsolètes, un nettoyage automatique est effectué en plusieurs fois où chaque itération est effectuée dans un court laps de temps. La durée maximale d'exécution du nettoyage de la table d'index est garantie afin de ne pas pénaliser le fonctionnement en temps réel du procédé d'échange de données. Cette durée maximale d'exécution du nettoyage peut être spécifiée par l'application de routage.

D'autre part, les fonctions de la bibliothèque informatique de stockage sont robustes à R-1 pannes simultanées, où R est le nombre de réplicas. En effet, les incohérences liées à des pannes sur les réplicas sont automatiquement réparées. Les fonctions de la bibliothèque logicielle de stockage sont résilientes même lorsque les pannes se répètent.

Les performances du système d'échange de données informatiques tournent autour d'environ 100 microsecondes par lecture d'un réplica sur une base de données KVS distante en utilisant du matériel informatique standard fonctionnant sur un noyau Linux. Par simple ajout de technologies Infiniband / RDMA (« Remote Direct Access Memory ») ou HSE / RoCE (« High Speed Ethernet » / « RDMA over Converged Enhanced Ethernet ») sur ces serveurs il est possible de réduire ce temps à environ dix microsecondes voire moins.

3.8 AUTRES AVANTAGES ET CARACTÉRISTIQUES DE L'INVENTION

Dans des variantes de mise en œuvre de l'invention, la flotte de clients informatiques peut être constituée de téléphones mobiles. Le serveur de routage connecté au système informatique en nuage de type RAM-Cloud ou SSD-Cloud permet d'échanger des données entre un téléphone mobile et un serveur de traitement. L'application de routage est alors adaptée aux caractéristiques techniques de la téléphonie mobile.

Dans des variantes de mise en œuvre de l'invention, les clients informatiques sont des bornes de jeux, des bornes de distribution de tickets, des box d'accès à Internet, des systèmes de collecte d'information (alarmes, compteurs d'eau ou d'énergie, badgeuses) ou des objets connectés. Les échanges de données informatiques peuvent être sécurisées ou non sécurisées.

3.9 REVENDICATIONS DE L'INVENTION

Le procédé intégré LC4 (300) ou procédé hystérétique probabiliste rétroactif intégré de lecture consolidée d'une paire dans le nuage de type RAM-Cloud, ledit procédé comprenant plusieurs constituants en interaction et/ou rétroaction, à savoir :

- La paramétrisation LC4 (302) ;
- La fonction ou procédé de lecture consolidée LC4 (340) comprenant :
    - Le prologue LC4 (340a) ;
    - Le corps LC4 (340b) ;
- Le procédé hystérétique (310) ;
- La table TableEtats (320) ;
- La fonction de probabilité LoiTransition (330).

3.10 SYNTHÈSE FORMELLE DE L'INVENTION

3.10.1 TITRE

PROCÉDÉ LOGICIEL ET SYSTÈME DE MOTEUR DE DONNÉES DISTRIBUÉES, TEMPS RÉEL, NON ADMINISTRÉ, EN NUAGE EN MÉMOIRE, POUR MAXIMISER LA PERFORMANCE ET LA FIABILITÉ DES ACCÈS AUX DONNÉES ET EN PARTICULIER POUR AFFAIBLIR LOCALEMENT LES EFFETS INHÉRENTS AU THÉORÈME DE CAP/BREWER LORS DES ACCÈS EN LECTURE, PAR L'INTERMÉDIAIRE D'UN INDEX MULTI-PLANAIRE À ÉVITEMENT DE CONCURRENCE ET D'UNE MÉTHODE "LC4" DE LECTURE CONSOLIDÉE DISTRIBUÉE COMPRENANT UN EFFET PROBABILISTE RÉTROACTIF HYSTÉRÉTIQUE.

3.10.2 ABRÉGÉ

L'invention concerne un procédé d'échange de données informatiques entre une flotte de clients informatiques et un parc de serveurs de traitement via un procédé de routage (301) comprenant un système de stockage de données en nuage en mémoire (134).

Le procédé de routage (301) comprend notamment une étape de lecture consolidée (340) des réplicas d'une information stockée dans le nuage (134).

L'étape de lecture consolidée (340) est comprise dans un procédé hystérétique probabiliste rétroactif intégré (300) comprenant également en interaction et/ou rétroaction, un procédé hystérétique (310), une table d'états (320), une fonction de probabilité (330).

Le procédé intégré (300) permet au procédé de routage (301) et à toute application accédant au nuage, de lire en temps réel les informations du nuage en maximisant la performance et/ou la fiabilité dans tous les cas d'usage, sous le contrôle de l'application et avec une granularité par requête.

3.10.3 FIGURES DE LA DESCRIPTION FORMELLE

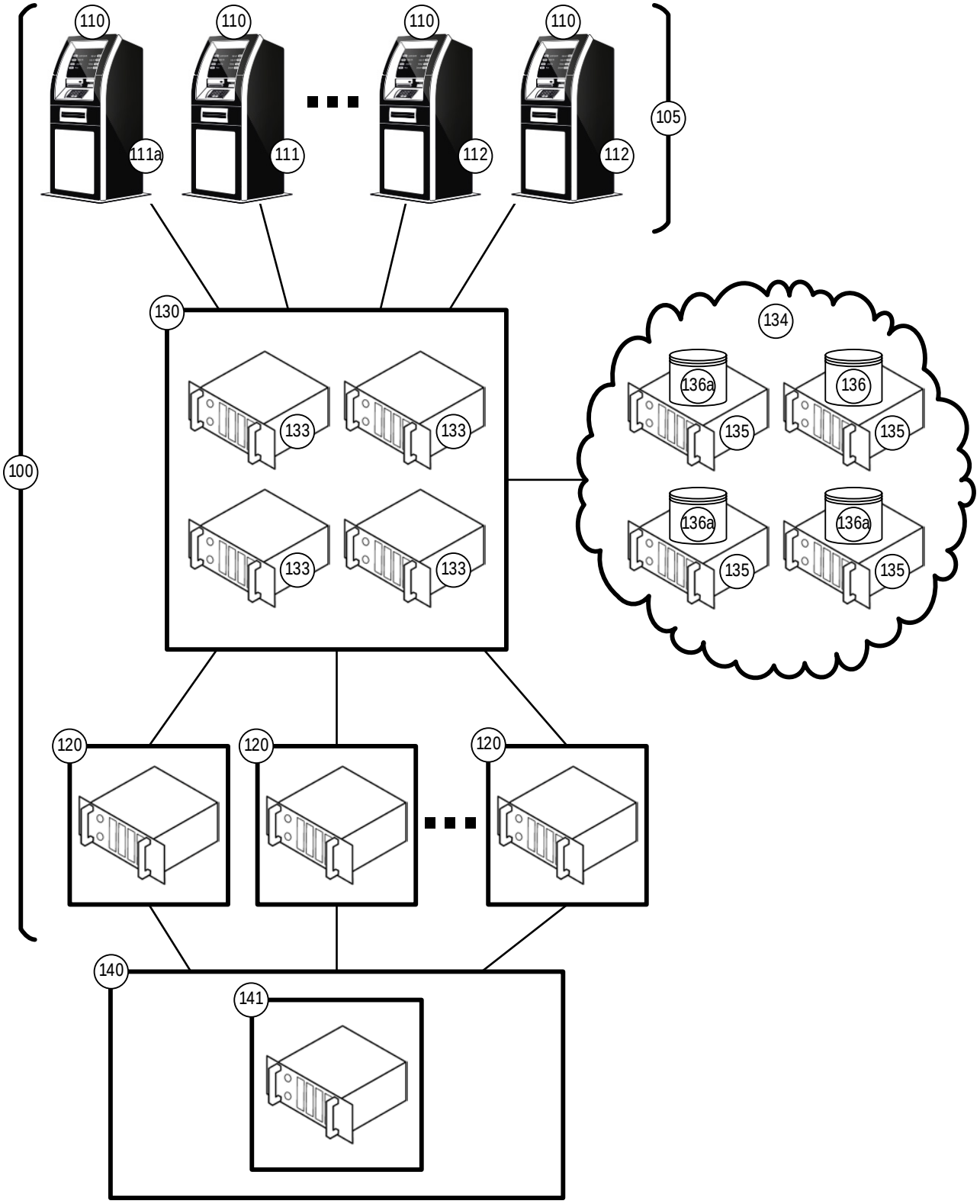

Figure Formelle 1

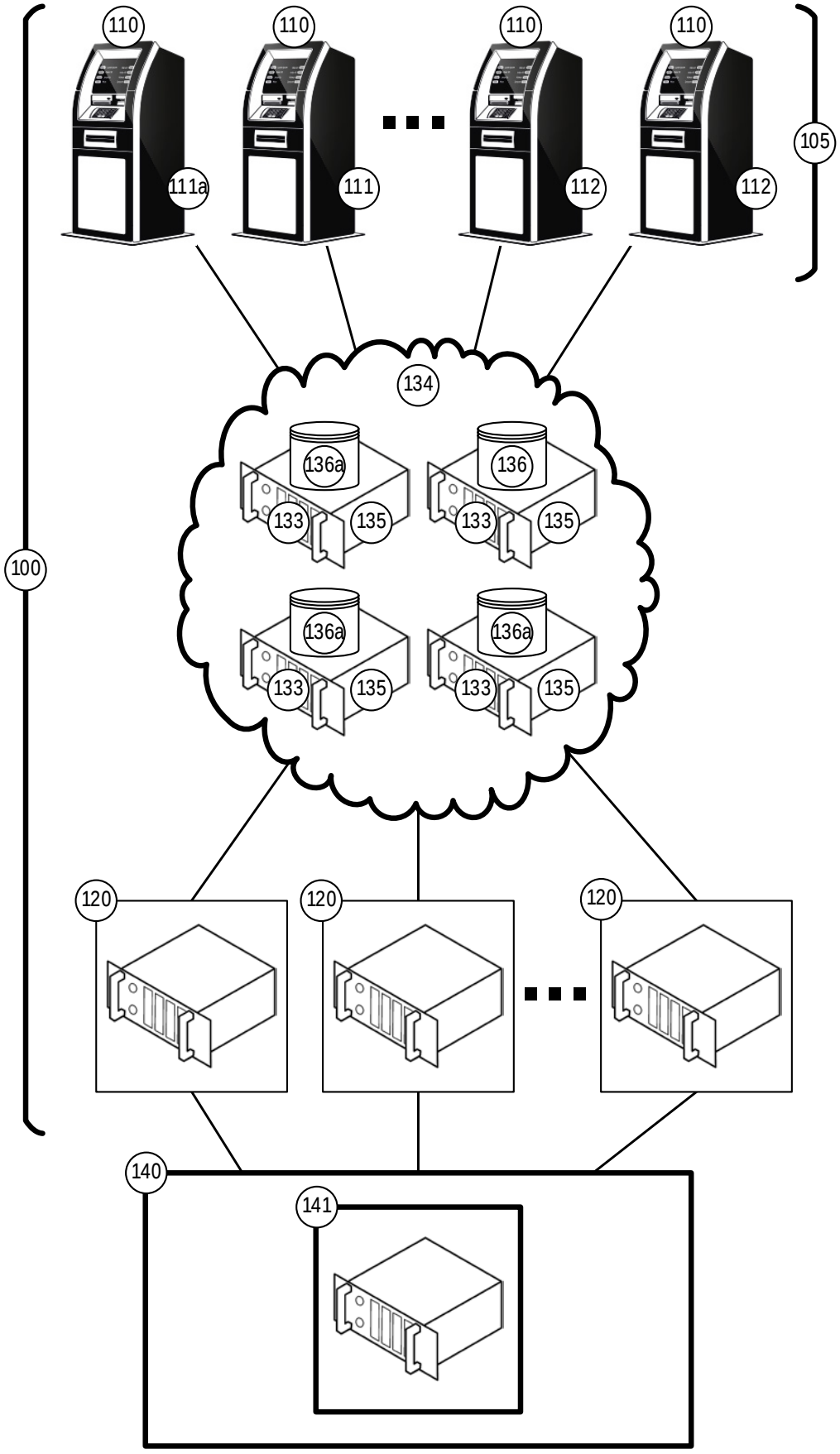

Figure Formelle 2

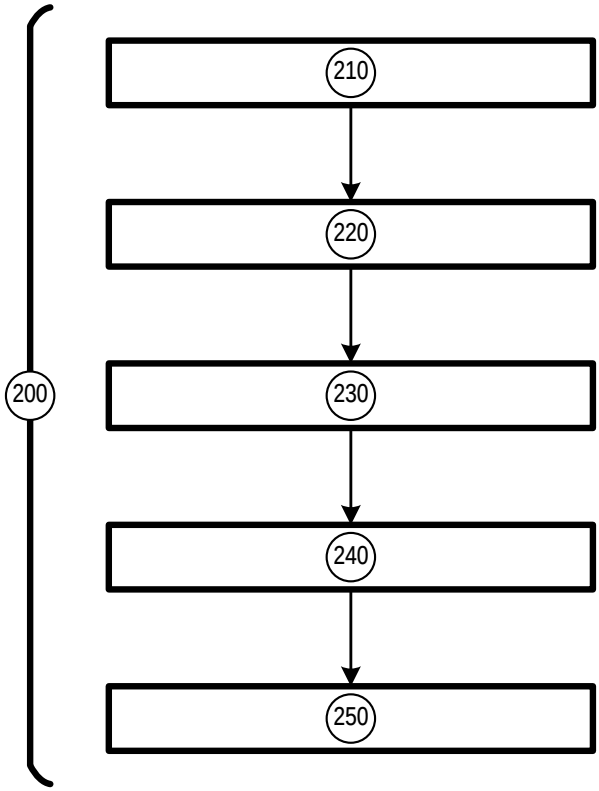

Figure Formelle 3

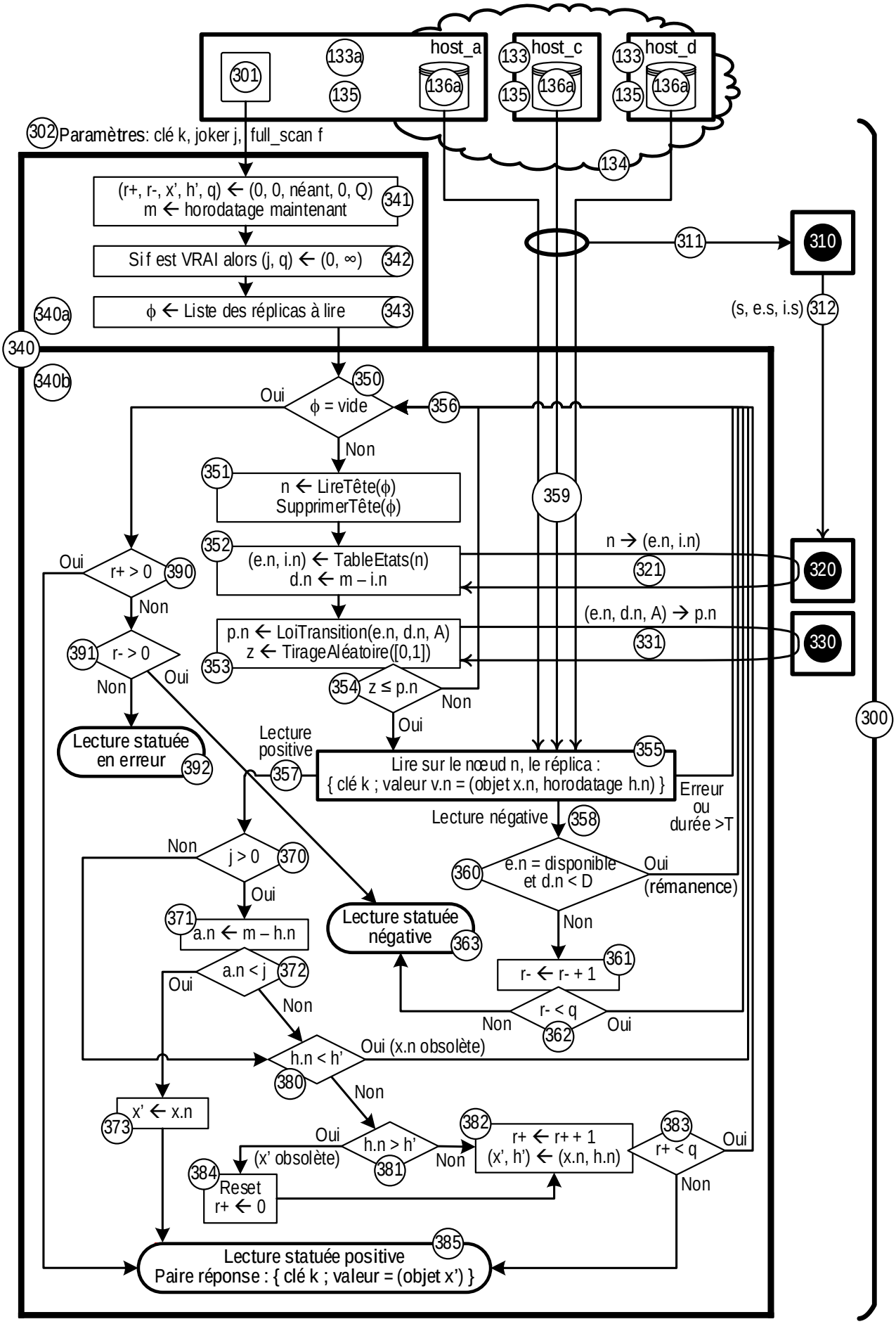

Figure Formelle 4

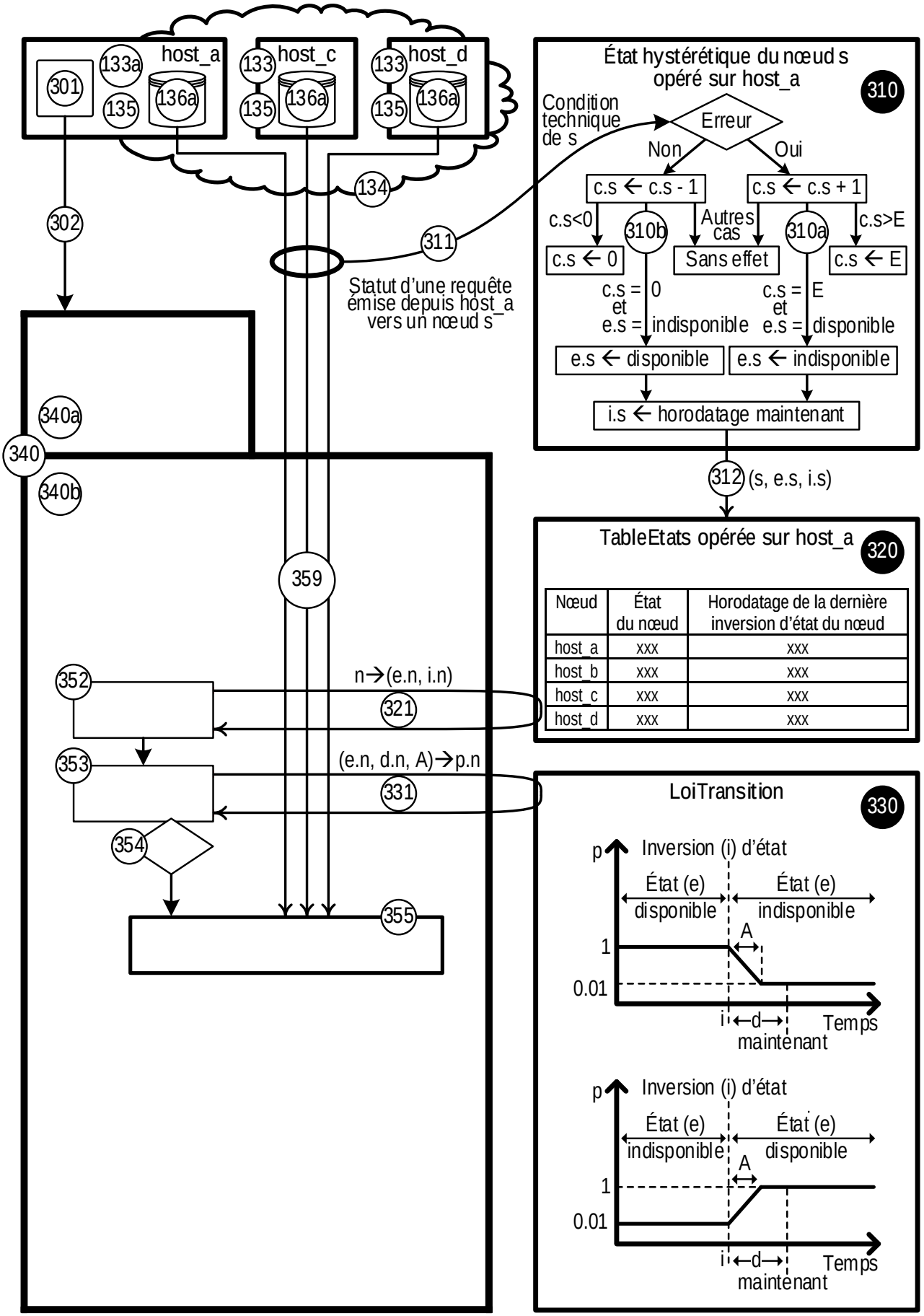

Figure Formelle 5