République Algérienne Démocratique et Populaire

Ministère de l'*Enseignement Supérieur* **et** *de la Recherche Scientifique*

Université Ibn Khaldoun de Tiaret

Faculté des Sciences et des Sciences de l'Ingénieur

Département d'Informatique

MÉMOIRE

Présenté pour l'obtention du diplôme de Magister

Filière :

Informatique

Option :

Système d'Information et de Connaissances

Réalisé par :

Mr. AID Lahcène

Étude de l'Apprentissage des Performances d'un Serveur Web par l'utilisation des Réseaux de Neurones Artificiels

Soutenu devant le jury composé de :

M. Y. DAHMANI	Président	M.C.A (Université de Tiaret)
M. A. BALLA	Examinateur	M.C.A (E.S.I d'Alger)
M. A.CHAIB	Examinateur	M.C.B (Université de Tiaret)
M. M. LOUDINI	Directeur du Mémoire	M.C.A (E.S.I d'Alger)
M. W.-K. HIDOUCI	Co-Directeur	M.C.A (E.S.I d'Alger)

2010

Remerciements

Je souhaite tout d'abord témoigner ma profonde reconnaissance envers Monsieur Malik Loudini et Monsieur Walid-Khaled Hidouci, Maîtres de Conférences "A" à l'École Nationale Supérieure d'Informatique d'Alger, pour avoir accepté de diriger ce travail de recherche et pour leurs précieux conseils et observations pour l'aboutissement et l'amélioration de notre travail.

Je suis très reconnaissant envers Monsieur Youcef Dahmani, Maître de Conférences "A" à l'université de Tiaret, qui me fait l'honneur de présider le jury de cette soutenance.

J'exprime toute ma gratitude à Monsieur Amar Balla, Maître de Conférences A à l'École Nationale Supérieure d'Informatique d'Alger, pour avoir accepté de juger ce travail en qualité d'examinateur et faire le déplacement pour faire partie de l'honorable jury.

J'adresse mes sincères remerciements à Monsieur Abdelkader Chaib, Maître de Conférences "B" à l'université de Tiaret, pour l'intérêt qu'il porte à mon travail, et pour avoir accepté la charge d'examiner le travail réalisé.

Mes vifs remerciements vont aussi à Mr. Youcef Meslem pour son accueil au sein du centre de calcul de l'université de Tiaret et son aide administrative.

Je remercie chaleureusement Mr. Abdelkader Maatoug, pour ses conseils et son aide qui ont bien contribué à l'avancement de ce travail.

Dédicaces

Je dédie ce travail

A

Mes chers parents, qui ont toujours été là pour moi et qui m'ont donné et me donnent un magnifique modèle de persévérance.

A

Mon épouse, pour avoir accepté tant de sacrifices durant les années de magister, pour son encouragement au quotidien.

A

Toute ma famille

A

Mes Amis

Résumé

Dans notre travail, nous nous intéressons à l'amélioration de la qualité de service (QoS) des serveurs web. Pour cela, nous proposons un modèle neuronal feedforward capable de prédire les critères de qualité de service : le temps moyen de réponse, le pourcentage de rejet et le débit du serveur web en fonction des paramètres du serveur Apache, du noyau du système Linux et du trafic d'arrivée. Le modèle sert à contrôler l'admission des requêtes sur la base des critères de qualité de service désirés. Il est, ensuite, utilisé pour la surveillance des performances du serveur web.

Mots clé : Serveur web, qualité de service, réseaux de neurones, modélisation mathématique, contrôle d'admission.

Abstract

In our work, we are interested in the improvement of quality of service (QoS) of the web servers. For that, we propose a feed-forward neural network model able to predict the quality of service criteria: average response time, blocking probability and throughput of web server based on the parameters of the Apache server, the core of the Linux system and arrival traffic. The model is used to perform admission control of the incoming requests on the basis of the desired criteria. It is, then, used for the monitoring of the web server performances.

Keywords**:** Web server, quality of service, neural networks, mathematical modelling, admission control.

SOMMAIRE

Chapitre I

Les serveurs web

Chapitre II

Etude et modélisation mathématique du serveur web Apache

Chapitre III

Réseaux de neurones

Chapitre IV

Mise en œuvre

Liste des figures

Liste des tableaux

Liste des acronymes

b_j : biais de neurone j de la couche k

HTTP : Hypertext Transfer Protocol.

MPMs : Multi-Processing Modules.

Qos : Quality of service

Qds : Qualité de service.

RNs : Réseau de neurones.

TCP : Transmission Control Protocol

V_j^k : La sortie du neurone j de la couche k.

V_j^0 : Entrée du neurone j de la couche d'entrée.

W_{ji}^k : Connexion entre le neurone i de la couche k-1 et le neurone j de la couche k.

δ_i^k : Gradient de l'erreur faite sur le neurone i dans la couche k.

Introduction générale

L'explosion de l'utilisation de l'Internet crée un grand défi pour les prestataires de services Internet et les opérateurs réseaux. A une heure de pointe, l'accès à un document ou la soumission d'une requête à un moteur de recherche peut prendre un temps considérable qui constitue un frein à l'utilisation interactive de l'Internet.

Un des soucis principaux des administrateurs des serveurs web est de proposer un service rapide et fiable qui satisfait leurs clients. Or, l'immense succès du web rend cette tâche difficile à accomplir. Il faut en effet être capable de fournir un service efficace à un instant donné, vu que le nombre et la nature de requêtes à traiter simultanément sont imprévisibles et rapidement variables.

A cet effet, il est important de développer des modèles permettant de planifier la capacité et de contrôler les performances d'un serveur web.

C'est dans ce cadre que se situe notre travail, qui vise à utiliser les capacités d'apprentissage et de généralisation des réseaux de neurones pour apprendre les critères de qualité de service à partir d'observations du système. Il s'agit de construire un modèle prédictif, c'est-à-dire destiné à estimer les critères de performance : débit, temps moyen de réponse, taux de rejet.

Cette prédiction permet au serveur de gérer au mieux les demandes des clients et d'éviter de s'écrouler en cas de surcharge.

Organisation du mémoire

Ce mémoire est organisé en quatre chapitres de la façon suivante :

Le premier chapitre présente les différents concepts liés aux serveurs web, suivi d'une description des différents types d'architectures des serveurs web. Ce chapitre décrit aussi le fonctionnement général d'un serveur web et sa qualité de service. Ensuite, il présente les approches existantes modélisant les serveurs web.

Le deuxième chapitre se divise en deux parties. La première décrit le fonctionnement interne du serveur web Apache et les paramètres effectifs de performance de ce dernier. La deuxième est dédiée à la modélisation du serveur Apache par file d'attente.

Le troisième chapitre est consacré à une présentation des réseaux de neurones artificiels.

Le dernier chapitre présente notre modèle neuronal. Celui-ci est capable de prédire les critères de performance du serveur web Apache. Ensuite il présente l'utilisation du modèle pour le contrôle et la surveillance des performances de ce dernier.

Enfin, une conclusion générale couronnant les travaux réalisés et présentés dans le cadre de ce mémoire est donnée. Elle inclura, aussi, divers axes de perspectives envisagées.

Chapitre I

Les serveurs web

I.1 Introduction

Les utilisateurs du web disposent des logiciels de navigation installés sur leur ordinateur local, « *le système client* ». Ce logiciel récupère l'information sur d'autres ordinateurs appelés « *serveurs* », sous forme de documents ou images par exemple.

Le succès du web dépend de la création d'information intéressantes, utiles ou encore exploitables. La vitesse, l'efficacité ainsi que la commodité de mise à disposition de l'information à l'utilisateur du web revêtent également une grande importance. Par conséquent, nous somme amenés à examiner les mécanismes d'amélioration des performances de qualité de service (QdS) des serveurs web.

Ce chapitre présente un état de l'art sur le web, le protocole TCP, le protocole HTTP, et les serveurs web.

Nous tenons à justifier dans ce qui va suivre l'utilisation de certains termes anglo-saxons cela est dû à l'absence, à notre connaissance, de termes français équivalents

I.2 Le World Wide Web

Le world wide web (souvent désigné par son acronyme WWW), ou plus simplement le web, est une toile d'araignée de serveurs d'informations reliés les uns aux autres par des liens physiques (le réseau matériel, Internet) et des liens logiques (les liens hypertextes). Grâce à sa grande convivialité, le web est à l'origine du succès populaire considérable de l'Internet.

Le web a été initialement développé par Tim Berners-Lee et Robert Cailliau dans les laboratoires du « CERN », à Genève en 1990, afin de donner aux physiciens des particules du CERN une infrastructure qui leur permet de partager l'information. Comme ceux-ci étaient situés dans différentes organisations et utilisaient une grande variété de systèmes informatiques et logiciels (notamment traitement de texte). Le Word Wide Web a été développé suivant une architecture client-serveur, qui assure une portabilité multiplateforme.

Le web est basé sur trois mécanismes pour rendre disponible ses ressources :

- Un schéma uniforme pour adresser les serveurs (i.e. URL)
- Des protocoles pour accéder aux ressources (e.g. HTTP)
- Un format pour naviguer entre les ressources (e.g. HTML)

I.3 Définition d'un Serveur web

Un serveur web est un logiciel servant des requêtes respectant le protocole de communication client-serveur HTTP. Il a été développé pour le world wide web.

La formule suivante caractérise un serveur de manière schématique [YEA 96] :

Serveur web = *Une plate forme* + du *logiciel* + *de l'information*

Un serveur web doit être un ordinateur connecté à internet, muni d'un logiciel système permettant son fonctionnement et permettant la connexion à d'autres systèmes internet. Le matériel, son système d'exploitation, et le logiciel réseau forment la « plate-forme » informatique. La deuxième composante d'un serveur web et tout simplement le logiciel serveur lui-même. C'est le « logiciel ». Et par-dessus tout, un serveur web doit avoir quelque chose à servir, sans information à mettre à disposition, un serveur web n'est pas d'une grande utilité.

D'après une statistique réalisée par Netcraft [NET 10], il existe actuellement plus de 255 millions de serveurs web repartis dans le monde, la plus grande partie utilisant Apache (Figure I.1). (Voir Annexe A pour plus de détail).

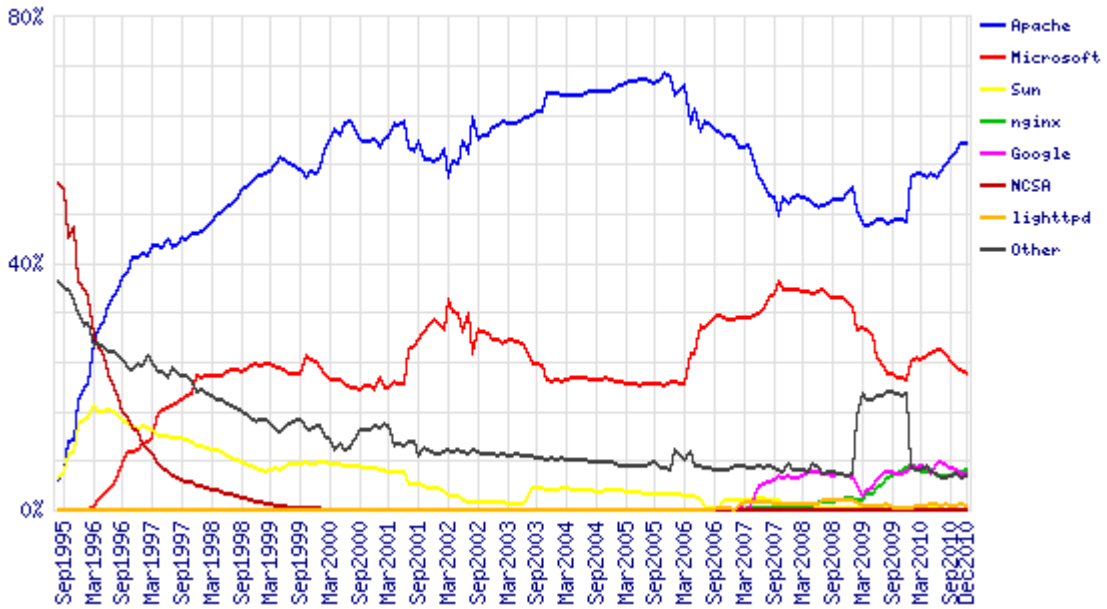


Figure I.1 : La distribution des serveurs web sur Internet en pourcentage de marché **(**[NET 10]- Décembre 2010)

Il est impossible de comprendre le web sans comprendre HTTP. Or, celui-ci est basé sur le protocole TCP. Pour cela, nous allons présenter le modèle TCP/IP suivi d'une description des deux protocoles TCP et HTTP.

I.4 Modèle TCP/IP

Le modèle TCP/IP est utilisé pour décrire et préciser le flux d'informations entre une source et une destination. il comporte les quatre couches suivantes [COM 98]:

- ✓ La couche application ;
- ✓ La couche transport ;
- ✓ La couche Internet ;
- ✓ La couche d'accès au réseau (Figure I.2).

Une couche ne définit pas un protocole, elle délimite un service qui peut être réalisé par plusieurs protocoles de différentes origines.

Un protocole de communications de données est un ensemble de règles, ou convention, qui détermine le format et la transmission des données.

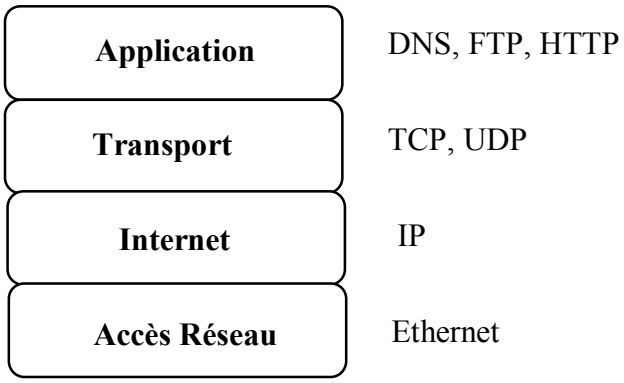


Figure I.2 : Modèle TCP/IP et les protocoles utilisés dans chaque couche

a. La couche Application

La représentation, le code et le contrôle du dialogue sont traités au niveau de cette couche. Une application interagit avec les protocoles de niveau transport pour émettre et recevoir des données.

b. La couche Transport

Le rôle principal de la couche transport est d'acheminer et de contrôler le flux d'informations de la source à la destination, de manière fiable et précise. L'un de ses protocoles, TCP, fournit d'excellents moyens de créer des communications réseau fiables.

c. La couche Internet

Elle gère les communications de machine à machine par la détermination du meilleur chemin. Le protocole principal de cette couche est : *IP* (Internet Protocol) qui permet à tout ordinateur de communiquer en tout temps et en tout lieu.

d. La couche Accès Réseau

Cette couche concerne tous les composants, à la fois physiques et logiques, qui sont nécessaires pour créer une liaison physique.
Deux concepts sont liés à la connexion entre ordinateurs à savoir les ports et les sockets.

- **Les Ports**

« Le Port » est le nom donné à une connexion logique entre le protocole *IP* et un processus de niveau plus élevé tel que HTTP. Tout hôte est capable de déterminer le numéro de port et les attributions de processus de tout ordinateur auquel il se connecte en interrogent le mappeur de ports de l'ordinateur. Le mappeur des ports est un processus dynamique qui assigne ou attribue le numéro de port suivant disponible à un processus.

- **Les Sockets**

Les sockets sont ce qu'on appelle une API (Application Program Interface) c'est-à-dire une interface entre les programmes d'applications et la couche transport, comme TCP ou UDP. En d'autres termes, une socket est un point de communication par lequel un processus peut émettre ou recevoir des données. Une socket comprend le couple (numéro de port, adresse IP).

I.5 Le protocole TCP

Le protocole TCP (Transmission Control Protocol - protocole de contrôle de transmission) est orienté connexion. Une connexion doit par conséquent être établie avant le début du transfert des données. TCP utilise des numéros de port pour transmettre les informations aux couches supérieures. Ces numéros servent à distinguer les différentes conversations qui circulent simultanément sur le réseau. Il est possible que sur un même ordinateur plusieurs connexions partagent un même numéro de port (e.g le port 80 pour les serveurs HTTP).

I.5.1 Fonctionnement de base de TCP

TCP effectue trois tâches dans l'ordre. La première, est l'établissement d'une connexion ; il s'agit d'établir une liaison de communication avec un hôte distant. L'opération suivante est la transmission effective des données que les deux parties concernées par la liaison de communication jugent nécessaires de s'échanger. Enfin, à la fin du transfert de données ou à la suite d'un acte délibéré des deux parties connectées, la connexion est coupée.

- **Etablissement d'une connexion TCP**

Pour établir une connexion, TCP utilise un processus en trois temps.

1- Le client envoie une demande de connexion, sous la forme d'un paquet SYN, au serveur. Le paquet SYN est employé pour synchroniser le numéro de séquence entre le client et le serveur. Par exemple ce premier SYN pourrait contenir le numéro 1.

2- Le serveur reçoit la demande et envoie une réponse SYN et un accusé de réception ACK (*acknowledgement*) au client. ACK contiendra le prochain numéro de séquence que le serveur compte recevoir, dans notre exemple le numéro 2 (figure I.3). Le paquet SYN contiendra le numéro de séquence initial pour le serveur, par exemple 10.

3- Le client envoie alors une réponse ACK au serveur. ACK contiendra le prochain numéro de séquence que le client compte recevoir du serveur, dans notre exemple 11 (figure I.3).

La connexion est maintenant établie, les données peuvent être transférées dans les deux sens.

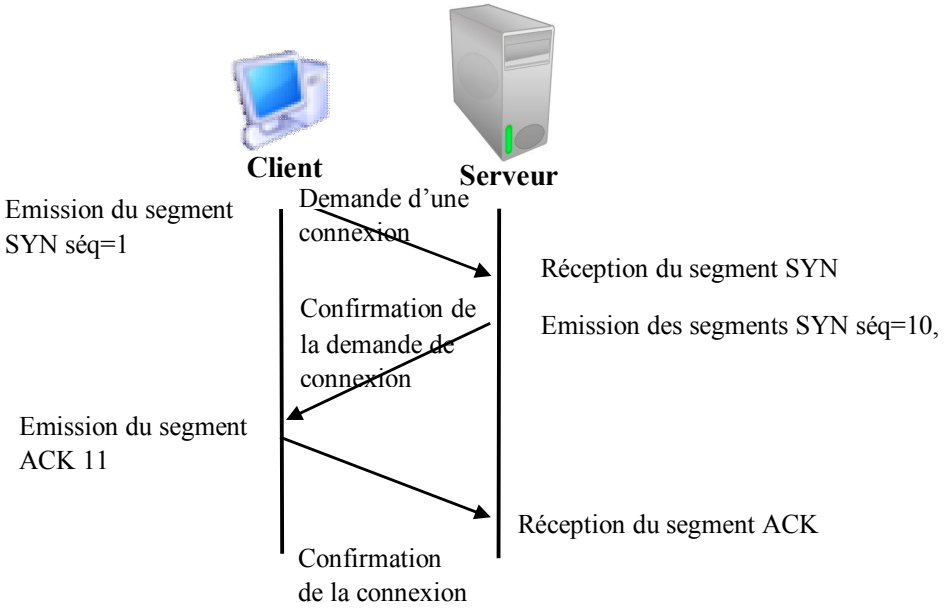


Figure I.3 : Séquence des messages échangés dans le processus en trois temps

- **Etats de connexion** [COM 98] :

TCP possède dix états de connexion de base énumérés dans le tableau 1.1. Ils sont classés selon un ordre qui rend compte du cycle de vie d'une connexion. En effet, elle progresse du début à la fin de sa vie en passant successivement par le premier état mentionné, puis le second, etc.

LISTEN
SYN SENT
SYN RECEIVED
ESTABLISHED
FIN WAIT 1
FIN WAIT 2
CLOSE WAIT
CLOSING
LAST ACK
TIME WAIT

Tableau 1.1 : Etats de connexion

LISTEN: est l'état d'attente d'une demande de connexion provenant d'une autre TCP distant.

SYN SENT: est l'état d'attente d'une demande de connexion appariée (en retour ou en écho) d'un TCP distant après l'envoi d'une première demande de connexion.

SYN RECEIVED: est l'état d'attente d'une confirmation de demande de connexion et d'un accusé de réception émis par un TCP distant après réception et envoi d'une demande de connexion.

ESTABLISHED : est l'état d'une connexion établie entre deux TCP. Dans cet état, les données reçues sont transmises au processus applicatif.

FIN WAIT 1 : est l'état d'attente d'une demande de terminaison de connexion provenant du TCP distant ou de l'accusé de réception émis par un TCP distant suite à la demande de fin de connexion provenant du TCP local.

FIN WAIT 2 : est l'état d'attente d'une demande de connexion provenant d'un TCP distant.

CLOSE WAIT : est l'état d'attente d'une demande de fin de connexion provenant du TCP local.

CLOSING : est l'état d'attente d'un accusé de réception d'une demande de fin de connexion provenant du TCP distant.

LAST ACK : est l'état d'attente de l'accusé de réception d'une demande de fin de connexion provenant du TCP distant.

TIME WAIT : est l'état d'attente d'un délai suffisant pour garantir que le TCP distant a reçu l'accusé de réception de sa demande de fin de connexion.

I.5.2 Temps d'aller retour

Le temps d'aller retour (RTT pour Round-trip time) est le temps mis par un client pour récupérer un document. Ce temps comprend les étapes : ouverture de la connexion réseau, envoi par le client de la requête au serveur, traitement de la requête, et le retour de la réponse au client.

Le RTT est propre à chaque client, diffère d'une requête à l'autre, et subit l'influence de beaucoup de paramètres en dehors du serveur web lui-même.

I.6 Le protocole HTTP

Un autre composant clé du web est le protocole HTTP [BER 96, FIE 99] (HyperText Transfer Protocol) **.**Celui-ci représente un ensemble de règles relativement simples et conçues pour être utilisées dans le cadre des systèmes hypermédia distribués en réseau.

HTTP est utilisé pour communiquer entre les clients et les serveurs web. L'histoire du standard HTTP a connu trois versions (0.9, 1.0 et 1.1), chacune d'entre elles apportant des améliorations aux versions antérieures, tout en gardant les standards déjà en place [SCU 01].

Toutes les transactions HTTP respectent le même format, Les requêtes et les réponses contiennent trois parties (Figure I.4) :

- Une ligne de début indiquant s'il s'agit d'une requête ou d'une réponse.
- Une section d'en-têtes.
- Le corps de la requête ou de la réponse.

I.6.1 Fonctionnement du protocole HTTP

HTTP définit un mode simple de conservation selon le modèle requête/réponse. La figure I.4 permet de mieux comprendre les transactions entre le client et le serveur.

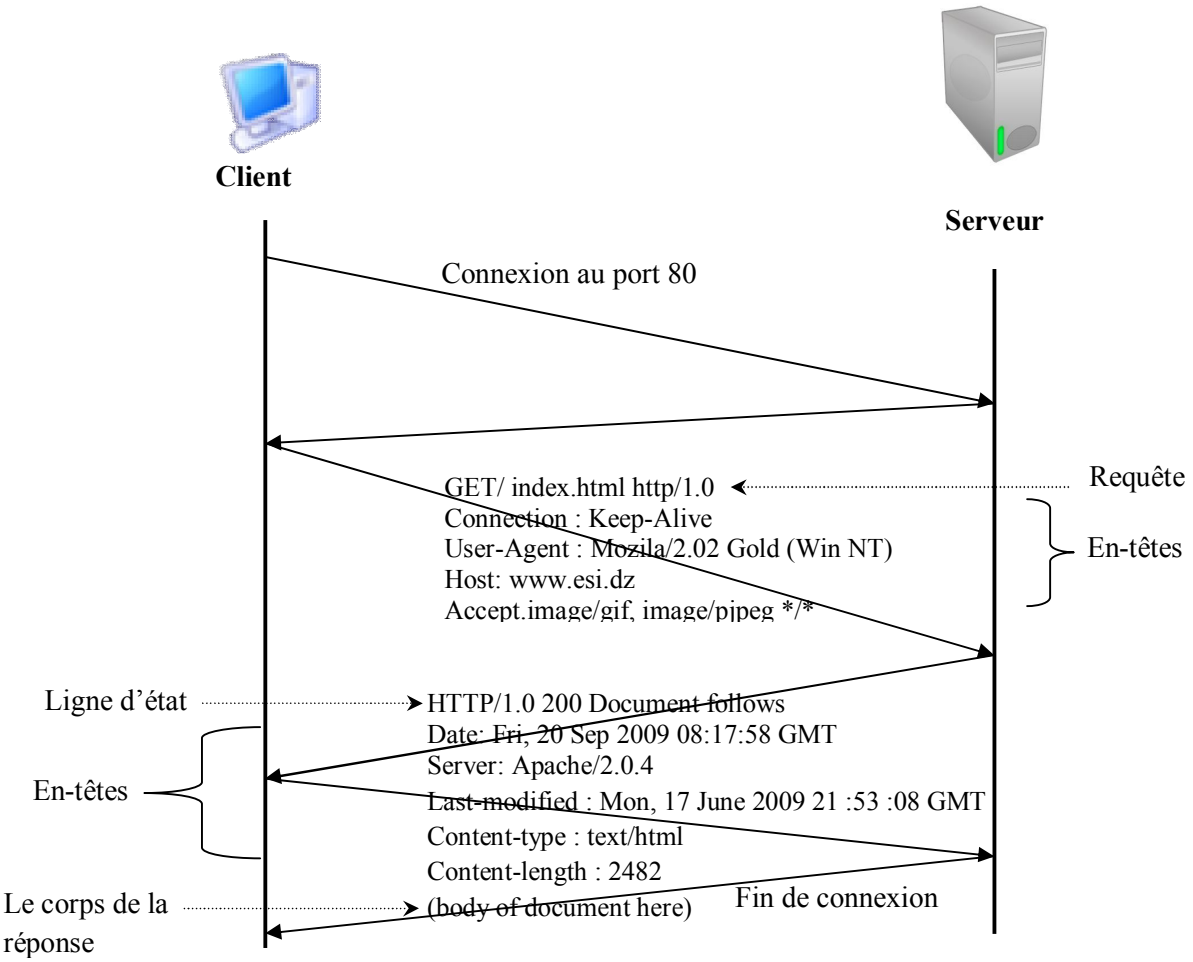


Figure I.4 : Un exemple de transaction HTTP/TCP

Connexion : La connexion est initiée par le client via le protocole TCP (voir section I.5.1) en mentionnant le nom du serveur ou son numéro IP, ainsi que le numéro du port spécifié dans l'adresse. Dans le cas où ce dernier n'est pas précisé, le port 80 est utilisé par défaut.

Requête : La requête envoyée par le client est constituée de 4 sections (figure I.4) :

1) Une ligne de requête, précisant la méthode utilisée, une ressource (un fichier) ainsi que la version HTTP utilisée, généralement HTTP/1.1

   Les trois principales méthodes sont GET, POST et HEAD :

   GET : demande d'un document.

   HEAD : demande de l'en-tête (HTTP) d'un document

   POST : dépose d'information sur le serveur (e.g un formulaire).

2) Des en-têtes, un par ligne, qui servent à apporter des informations facultatives au serveur : les langues acceptées par le client, le type du navigateur, les cookies, les types de compression acceptées, etc.

3) Une ligne vide, pour signaler la fin des en-têtes.
4) Un corps de requête optionnel, contenant par exemple un formulaire

Réponse : La réponse d'un serveur est également constituée de 4 sections :

1) Une ligne d'état contenant la version du protocole utilisé par le serveur, ainsi qu'un code de retour (voir Annexe B).

2) Des en-têtes, avec le même rôle que pour la requête, comme "type du contenu", qui décrit le type de l'objet qui est donné, et "longueur du contenu", qui indique la longueur du document.

3) Une ligne vide, pour signaler la fin des en-têtes.
4) Le corps de la réponse

Déconnexion : La connexion TCP-IP est coupée par le serveur quand tout le document a été transmis. Le client peut avorter le transfert en coupant lui-même la connexion avant la fin du transfert, dans ce cas le serveur n'enregistre aucune erreur.

I.6.2 HTTP 1.0 versus HTTP 1.1

Une page web est constituée d'un ensemble d'objets. Pour récupérer la page, le navigateur doit faire plusieurs requêtes au serveur. Il fait une requête pour chaque objet intégré dans la page (image, script, etc.). Pour cela, le mécanisme de gestion de ces connexions est très important pour la performance d'un navigateur web.

En utilisant le protocole HTTP/1.0, pour chaque nouvelle requête le client réalise une nouvelle connexion TCP avec le serveur. Mais pour établir une connexion TCP, un nombre de paquets supplémentaires est échangé, et cela prend un temps conséquent. Ce problème est connu sous le nom de "démarrage lent TCP" (TCP slow-start).Si pour chaque requête on doit attendre que la connexion TCP se réalise, les performances de transfert seront réduites d'une façon importante.

La version 1.1 du protocole HTTP permet la réutilisation d'une connexion TCP déjà ouverte pour les autres requêtes adressées au même serveur (connexion persistante). L'utilisation des connexions persistantes a comme effet un important gain de

performance, permettant le transfert de plusieurs objets sur la même connexion TCP.

I.7 Typage des documents

Le web avait initialement pour vocation de rendre accessible des documents hétérogènes. Pour cela, le protocole HTTP utilise un sous-ensemble de MIME (MIME pour *Multipurpose Internet Mail Extensions)*, qui permet d'envoyer des documents de différents types par courrier électronique. Le serveur indique dans l'entête de ses réponses le type du document au client. Ainsi, le MIME-type d'un document PDF est « application/pdf », alors que celui d'un document HTML est « text/html ».

I.8 Architecture des serveurs web

Les performances d'un serveur web dépendent en bonne partie de son architecture. On distingue cinq types d'architectures :

- ✓ **process-driven;**
- ✓ **Threaded;**
- ✓ **Event-driven;**
- ✓ **Hybride (Event-driven/ Threaded);**
- ✓ **In-kernel.**

Un serveur web *process-driven* [APA 10] lance des instances du programme (i.e. processus) *HTTP démon* offrants des services au lieu de communiquer directement avec l'utilisateur. L'architecture *Threaded* [IIS 10] désigne que dans le même processus, plusieurs fils d'exécution sont en cours de manière simultanée. Chaque thread est implémenté comme s'il était seul. Tandis que, l'architecture *Event-driven* (évènementielle) [ZEU 10] est organisée autour d'une boucle de contrôle et de traitants d'évènements. Le programme enregistre des traitants pour certains évènements. Lorsque la boucle de contrôle détecte l'arrivée d'un événement tel que : l'arrivée d'une nouvelle requête, une opération d'E/S, etc., elle exécute le traitant associé. Les opérations faites dans les traitants doivent être non bloquantes, comme il n'y a qu'un seul processus, pour que le serveur soit plus réactif.

Une architecture hybride [FLA 10] combine à la fois l'architecture threaded et Event-driven. Cependant, le problème de la disponibilité d'E/S asynchrones dans le système d'exploitation est résolu du fait que le traitant démarre un processus qui gère cette E/S et enregistre un nouveau traitant pour la fin de cette E/S. Ainsi, seul le processus gérant la communication avec le disque est bloqué en attente de la fin de l'opération.

Le serveur web in-kernel [KHT 10, TUX 10] est généralement conçu pour servir les demandes de pages statiques. Le serveur web est compilé comme un module de noyau du système d'exploitation, et doit être chargé dans la partie mémoire réservé au noyau , ainsi il peut servir les pages d'une manière plus rapide que les serveurs web classiques.

I.9 Fonctionnement d'un serveur web [YEA 96]

Le travail du serveur web est très simple : il reçoit des requêtes de la part des navigateurs web pour obtenir des documents à travers le réseau, les déchiffre pour déterminer le fichier à rechercher, trouve ce fichier s'il existe, et l'expédie au navigateur web en empruntant la connexion réseau.

Le fonctionnement d'un serveur web dépond de l'architecture qu'il implémente, et ce, pour traiter le plus de requêtes simultanément :

- ✓ En dupliquant le programme httpd pour chaque requête ;
- ✓ En découpant en processus léger (thread) le programme httpd ;
- ✓ En déléguant le travail à des programmes auxiliaires.

Dans le cas d'un programme serveur qui se duplique, chaque requête est traitée par une nouvelle copie du programme httpd. Le programme se sépare en deux copies identiques. L'original (le processus parent) retourne immédiatement à l'état d'attente d'une nouvelle requête (écoute un port jusqu'à ce que quelqu'un l'interroge). La nouvelle copie (processus fils) traite la requête. Lorsque le processus parent engendre le fils, celui-ci hérite d'une connexion réseau. De cette manière, chaque fils possède sa propre connexion sur laquelle il reçoit la requête qu'il est censé la satisfaire. Lorsque le processus fils achève de traiter la requête, il se termine et cesse d'exister à jamais. Le serveur web peut avoir de nombreuses copies du programme httpd en exécution simultanée, avec autant de connexions réseau actives en même temps. La copie principale engendre fils après fils au fur et à mesure de l'arrivée des requêtes. Chaque requête hérite une copie personnelle du programme, et chaque processus traite une requête unique.

Une seconde technique de traitement des requêtes simultanées consiste à programmer le serveur httpd de manière à ce qu'il exécute de multiples processus léger. Le serveur peut, par exemple, conserver une trace de l'avancement de plusieurs connexions, commutant de l'une à l'autre en cas de besoin. La première requête est traitée et la première partie des données est lue sur le disque. Plutôt que d'attendre la fin de cette opération relativement longue, le serveur mémorise son état courant (la requête, le port associé, et ce qui a déjà été réalisé jusqu'à présent) et entame la seconde requête. Lorsque celle-ci marque une pause, le serveur vérifie que la première requête a achevé sa lecture. Si tel est le cas, la

première requête reprend son cours jusqu'à une nouvelle pause, et ainsi de suite. En fin de traitement d'une requête, le serveur ferme les fichiers et connexions réseau et se trouve ainsi prêt à traiter une nouvelle requête.

La troisième technique consiste à concevoir le serveur httpd comme un ensemble de programmes coopérants. Une méthode consiste à faire effectuer la lecture des requêtes sur le réseau à un programme tandis que d'autres se chargent du reste.

I.10 La qualité de service

La qualité de service (QoS, Quality of Service en anglais) est définie comme «l'effet général de la performance du service qui détermine le degré de satisfaction d'un utilisateur du système» [UIT 94].

Cette satisfaction requiert l'opinion de l'utilisateur. Cependant, les temps de réponse jugés longs et les navigations difficiles sont les deux principaux inconvénients cités par les utilisateurs d'Internet. Après avoir attendu au delà d'un certain seuil de patience, il est fort possible que l'utilisateur soit frustré et quitte le site pour un autre plus rapide. Ainsi, Le serveur web doit assurer une bonne qualité de service pour ses clients.

La notion de « qualité de service » est en général traduite en termes de critères quantifiables de performance. Les critères plus importants pour un serveur sont [CAO 03, LIN 08] :

- **Le temps de réponse :** le temps de réponse d'une requête client est défini comme le temps nécessaire au serveur pour traiter cette requête.
- **Le débit (throughput) :** est le nombre de requêtes traités par une unité de temps.
- **La disponibilité :** est le pourcentage du temps dans lequel le système est disponible.

I.10.1 Temps de réponse

Le temps de réponse est étroitement lié à la charge du serveur, celle-ci est caractérisé d'une part par le nombre de clients qui essayent d'accéder concurremment au serveur, et d'autre part par la nature des requêtes soumises par les clients.

I.10.2 Débit (throughput)

Pour un serveur web, le nombre de requêtes reçues par seconde varie fortement. Pendant que les utilisateurs cliquent sur des ancres, il se produit un flux et un reflux de requêtes. Il en résulte des périodes où le nombre de requêtes qui arrivent au serveur est supérieur ou inférieur à la moyenne. Les systèmes serveurs ont des capacités de réponse à des

requêtes limitées. Certains systèmes peuvent s'effondrer en cas de surcharge. Il est préférable de connaître le nombre maximal des requêtes qui peuvent être traités par le système par seconde, plutôt que de s'en connaitre le nombre moyen. Le nombre maximal des requêtes représente le trafic que le serveur peut absorber lorsqu'il est sous la pression de demande.

I.10.3 Disponibilité

Une forte charge peut provoquer un écroulement du serveur et entraîner une indisponibilité du service. Quand un serveur web est surchargé, le temps de réponse devient long (figure I.5), se qui est traduit par une baisse de performances et une perte de service.
Les coûts engendrés par ce phénomène sont estimés à plus de 2 millions de dollars par heure pour les entreprises de télécommunication et les entreprises financières [MOU 01, NAS 08].

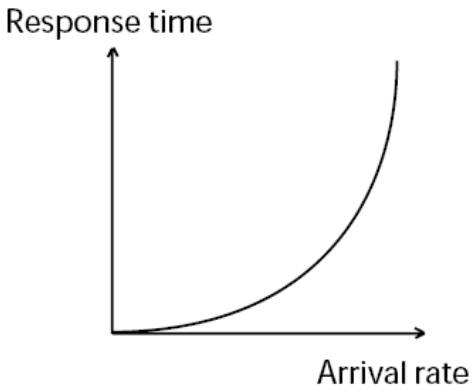


Figure I.5 : Rapport charge de serveur / Temps de réponse [AND 04]

Le temps de réponse tend vers l'infini quand la charge de serveur augmente.

I.11 Méthodes d'amélioration de la QdS d'un serveur web

Les serveurs web actuels sont capables de servir des millions de requêtes par jour, le temps moyen de réponse d'un serveur peut être long durant la période de pointe. Cette situation frustre les utilisateurs qui sont en interaction avec le serveur et déprécie ainsi la qualité de service qui est initialement liée à la configuration du serveur web [LOO 97, MAR 03], sera détaillé ultérieurement (voir chapitre II).

Certains serveurs implémentent le traitement simultané de plusieurs requêtes par la création des processus ou des threads, comme déjà mentionné, pour chaque connexion entrante. Or, la création de nouveaux processus peut s'accroître et peut ainsi entraîner une surcharge du serveur. D'où la nécessité d'un système de contrôle d'admission pour assurer un débit et un temps de réponse acceptable du serveur web.

I.11.1 Contrôle d'admission

C'est un mécanisme qui permet d'empêcher le serveur de s'écrouler quand la charge devient trop importante. Cela consiste à limiter le nombre de requêtes entrantes. Au delà de cette limite, les requêtes des clients sont rejetées (figure I.6). Par conséquent, le serveur web effectue un temps de réponse raisonnable pour les demandes acceptées.

Ce mécanisme peut être statique ou dynamique. Un mécanisme statique admet un nombre prédéfini de demandes, tandis qu'un mécanisme dynamique possède un contrôleur qui calcul périodiquement un nouveau taux d'admission selon les objectifs de contrôle.

Les décisions du contrôleur sont basées sur des mesures de certaines variables de contrôle, comme la longueur de la file d'attente, le taux d'occupation du processeur et le délai de traitement [AND 03].

L'objectif du contrôle est de maintenir les valeurs des variables à des valeurs optimales. Le choix des variables de contrôle est très important, pour cela, ils doivent êtres facilement mesurables.

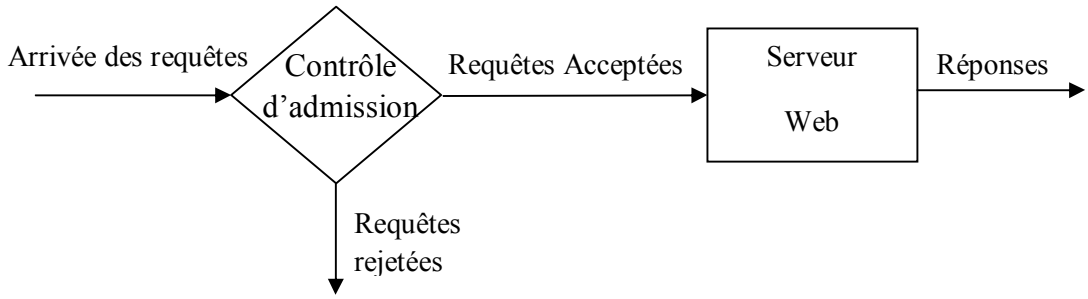


Figure I.6 : Mécanisme de contrôle d'admission

Le taux de rejet des requêtes clients est défini comme le rapport entre les requêtes rejetées à cause du contrôle d'admission et le nombre total de requêtes soumises au serveur. Un faible taux de rejet est préférable et cela reflète de la bonne disponibilité du service.

Peu de travaux ont été menés dans le domaine du contrôle d'admission. Un algorithme de contrôle d'admission appelé PACERS (Periodical Admission Control based on Estimation of Request rate and Service time) a été proposé par Chen et al [CHE 01] en utilisant deux files d'attente. La qualité de service a été assurée par une allocation périodique des ressources systèmes basée sur l'estimation du taux d'arrivée des requêtes et le temps de service requis pour les processus de priorité élevée.

Un mécanisme de contrôle d'admission a été conçu par Andersson et al [AND 03] en combinant un modèle de file d'attente avec des méthodes de la théorie du contrôle. Le but est

de contrôler la charge du CPU du serveur web. Un contrôleur PI (Proportionnel Intégral) a été conçu pour le serveur web Apache.

Elnikety et al [ELN 04] ont proposé un contrôleur d'admission appelé Gatekeeper qui identifie les différents types de requêtes (i.e servlets) et maintient en ligne le temps de service requis pour chaque requête à partir des mesures faites sur des exécutions précédentes. Connaissant la capacité du serveur web, le contrôleur d'admission peut maintenir la charge au dessus de cette capacité pour assurer un débit et un temps de réponse acceptables.

Fontaine & al [FON 07] proposent un mécanisme de contrôle d'admission à partir de l'apprentissage des performances d'un serveur web. Le Contrôle d'Admission fonctionne à base du temps moyen de réponse estimé par un réseau de neurone. L'admission des requêtes est décidée en comparant le temps de réponse estimé par le réseau de neurone avec celui fixé d'avance comme étant acceptable

I.11.2 Adaptation de contenu

L'adaptation de contenu est définit généralement comme le processus qui transforme un contenu de son état initial vers un état final afin de satisfaire un ensemble de contraintes. Les pages possédant un contenu lourd (image, vidéo,…) sont allégées pendant le chargement, afin d'améliorer la qualité de service.

Par exemple si une information très importante est publiée dans un site d'informations, et qu'un nombre important de visiteurs veulent accéder à ce site au même temps, on diminue la qualité de la page par la suppression des images et le contenu dynamique, pour éviter la surcharge du serveur web.

I.11.3 La répartition de charge (LOAD-BALANCING)

La répartition de charge (load-balancing en anglais, littéralement équilibrage de charge) est une technique pour distribuer les requêtes entre plusieurs serveurs groupés, de façon à optimiser les performances.

Le principe de base consiste à interposer entre les requêtes et le groupe de serveurs un dispositif (le répartiteur ou load-balancer) qui connaît l'état d'occupation de chaque serveur et qui est capable de diriger chaque requête vers le serveur le moins occupée, ou le plus facilement accessible.

La figure I.7 illustre ce mécanisme : le serveur web qui reçoit la requête répond en utilisant les nœuds de stockage reliés à lui par un système de fichiers distribué

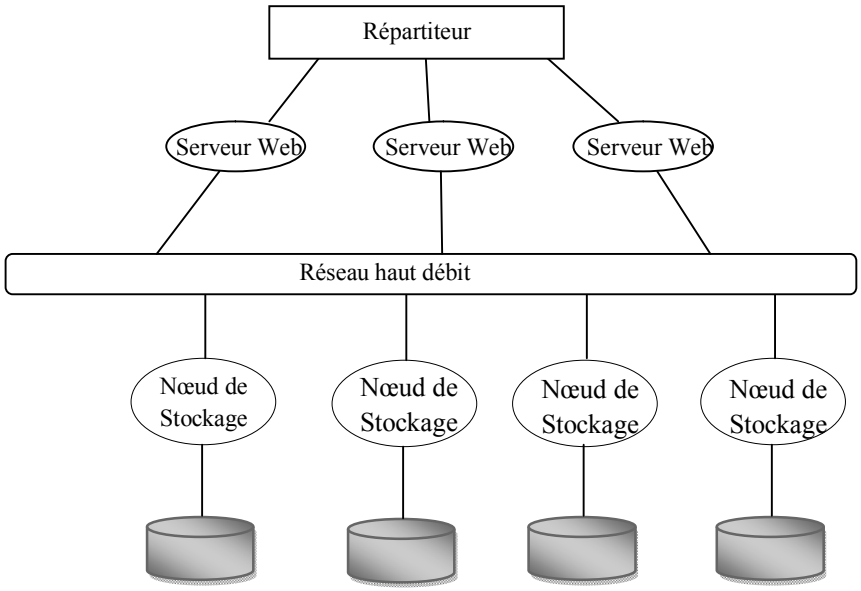


Figure I.7: Répartition de charge entre un groupe de serveurs web [SCU 01]

I.11.4 L'ordonnancement

C'est une autre technique, qui consiste à utiliser la notion de priorité dans le traitement des requêtes. Les requêtes sont divisées en deux classes : l'une pour la priorité supérieure, et l'autre pour la priorité inférieure.

L'ordonnancement dans le serveur web offre un temps de réponse acceptable pour les processus de priorité supérieure contrairement à ceux de priorité inférieure.

I.12 Modélisation des Serveurs web

I.12.1 La modélisation

Un *modèle* est définit comme une représentation théorique souvent simplifiée et relativement abstraite d'un système, capable de reproduire certains aspects de son comportement. C'est la résolution du modèle suivie du calcul de ses performances qui établit l'évaluation de performances du système [JAI 91].

I.12.2 Approches modélisant les serveurs web

Les modèles d'évaluation de performance sont utiles pour localiser les goulets d'étranglement responsables de l'accroissement indésirable des temps de réponse de certains serveurs web. Plusieurs facteurs affectent les performances : la vitesse du client, la bande passante client/serveur et la capacité du serveur lui-même [RAF 05].

Il existe des approches qui visent à modéliser le serveur web pour caractériser sa capacité : Le modèle linéaire à paramètre variant (LPV) est un type de modèle de système, dans lequel les coefficients dépendent d'un ou plusieurs paramètres variables dans le temps [QIN 07, SUN 08]. Les boites noires SISO (une entrée, une sortie) ou MIMO (plusieurs entrées, plusieurs sorties) [HEL 04, DIA 02] ; Le terme boîte noire est employé car seulement les entrées et les sorties du système doivent être connues. Le Modèle fluide [HEL 04] **:** une approximation de type fluide consiste à voir les variables d'état du système.

D'autre travaux [MEI 01, CAO 03, YAS 00, MIK 03] ont proposé des modèles de file d'attente M/M/1/K, M/G/1/K*PS, M/G/1/PS et MMPP/G/1/K*PS pour évaluer les performances d'un serveur web. Ils ont étudié l'impact sur les performances des paramètres tels que le trafic d'arrivée, la distribution de la taille des documents, la mémoire cache et le nombre maximal de connexions.

On s'intéresse à un modèle qui s'applique à des critères de performance tel que le temps de réponse et le taux de rejet, qui sont des critères de QdS directement perçus par les clients. En d'autres termes, un modèle qui permet de répondre à des questions comme : quel est le pourcentage de rejet ? quel est le temps moyen de réponse ?

La théorie des files d'attente permet d'obtenir des expressions analytiques (parfois exactes) de critères de performances sous des hypothèses relativement fortes. Cao et al. [CAO 03] ont montré qu'une simple file M/G/1/K*PS peut suffire à représenter correctement les performances d'un serveur web à condition que ses paramètres soient correctement calibrés. En effet, une fois calibrée, la file M/G/1/K*PS reproduit avec précision les débits, les temps de séjour et les taux de perte mesurés sur le serveur web à différents niveaux de charge. (Nous aborderons plus en détail ce modèle dans le chapitre 2)

I.13 Conclusion

Nous avons présenté dans ce chapitre les différents concepts liés aux serveurs web, suivi d'une description des différents types d'architectures des serveurs web. Nous avons mis en évidence le fonctionnement général d'un serveur web et sa qualité de service. Enfin, nous avons présenté les travaux existants modélisant les serveurs web. Cependant, ces travaux se limitent seulement à la modélisation et au dimensionnement du serveur web sans prendre en compte les paramètres du système d'exploitation sur lequel il s'exécute et les paramètres effectifs du serveur web. Donc, il est important de développer un modèle qui répond à ces exigences.

Nous avons choisi le modèle de file d'attente comme modèle de base pour construire notre modèle d'estimation des critères de performance du serveur web. Le chapitre suivant présente en détail le modèle choisi.

Chapitre II

Etude et modélisation mathématique du serveur web Apache

II.1 Introduction

Afin de modéliser un système il faut, tout d'abord, étudier son comportement. Pour cela, nous allons étudier le fonctionnement interne du serveur web Apache. On n'examinera pas tout le code source d'apache, on se focalisera sur le volet de l'amélioration des performances, et ce, pour construire la base d'apprentissage du Réseau de Neurones (chapitre 4).

Ce chapitre est organisé en deux parties. Dans la première partie nous présentons une étude du serveur web Apache. Tandis que la deuxième est consacrée à la modélisation mathématique du serveur Apache, notamment le modèle proposé par Cao et al [CAO 03].

II.2 Le serveur web Apache

II.2.1 Historique

Apache est un serveur web open-source basé sur le protocole HTTP, produit par *l'Apache Software Foundation* [APA 10]. Depuis 1996, Apache est devenu le server web le plus populaire dans le monde selon l'étude de *netcraft* [NET 10]. Et c'est pour ces raisons que nous avons choisi apache pour notre modélisation.

La première version d'Apache a été réalisée en 1994, elle est basée sur le serveur web NSCA httpd 1.3, ce dernier a été développé au sein de NSCA (**N**ational **C**enter for **S**upercomputing **A**pplications) à l'Université d'Illinois, par Rob McCool. Le nom *Apache* vient de *a patchy server*, soit « un serveur rafistolé ».

II.2.2 Fonctionnement d'apache

Le serveur HTTP Apache est un programme modulaire. Mis à part quelques modules directement intégrés dans le programme binaire httpd, l'administrateur peut choisir les fonctionnalités qu'il souhaite en activant des modules. Apache s'exécute en arrière plan, sur une très grande variété de plateformes, offrant ses services aux autres applications qui communiquent avec lui, par exemple le navigateur web [WAI 00]. Apache 2.0 étend cette conception modulaire aux fonctions les plus élémentaires d'un serveur web. Certains Modules Multi-Processus (MPMs) sont responsables de l'association aux ports réseau de la machine, acceptent les requêtes, et se chargent de répartir ces dernières entre les différents processus enfants [WAI 04]. Les MPMs permettent à Apache de traiter plusieurs requêtes simultanément, ils doivent être choisis à la configuration, et compilés avec le serveur.

Le tableau suivant fournit la liste des MPMs par défaut pour divers systèmes d'exploitation.

Unix	Prefork
Windows	mpm_winnt
BeOS	Beos
Netware	mpm_netware
OS/2	mpmt _os2

Tableau 2.1 : Liste MPMs [WAI 04]

II.2.2.1 Le mode Prefork

Sous UNIX, Apache est un multiprocessus : Lors de son démarrage, un processus père (*Master Server*) est lancé, il lit le fichier de configuration (*httpd.conf*), et il crée, par des appels systèmes *fork()*, des processus enfants (*child Server*) afin de traiter les requêtes adressées au serveur web. Chaque nouveau processus créé de cette façon est une copie exacte du processus Apache père.

Pre-fork ou La préduplication crée une réserve de processus fils à l'avance, même en absence des requêtes, prêtes à gérer les requêtes clients entrantes.

La figure II.1 illustre ce mécanisme [GRO 04] :

Le processus père a en charge :

- ✓ Le démarrage et l'initialisation du serveur : lecture de la configuration, et création des ressources.
- ✓ La création des processus fils et la gestion de cet ensemble selon le modèle *MPMs* choisi.
- ✓ L'attente des signaux de contrôle pour le pilotage du serveur (*stop, restart, graceful).*

Les processus fils se chargent :

- ✓ De l'écoute sur les *sockets* réseau et l'acceptation de nouvelles connexions.
- ✓ Du traitement des requêtes : interprétation du protocole *HTTP,* recherche des *URL* demandées, vérifications des droits d'accès,…
- ✓ Du traitement et de la création du contenu à retourner au client.

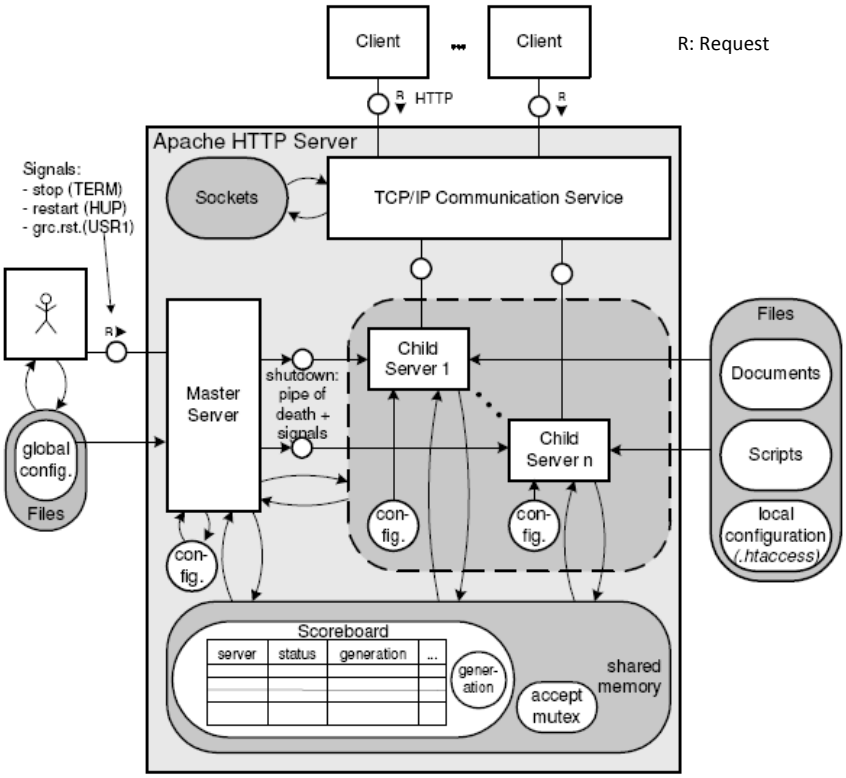


Figure II.1 : Mode prefork pour Apache 2 [GRO 04]

II.2.2.2 Le mode Worker

En mode Worker, Apache lance des threads (voir section I.9) qui se chargent de gérer les demandes entrantes. Contrairement au mode Prefork, le mode worker est un mode plus préemptif dans lequel le processus père prépare les ressources pour ses threads.

Nous avons utilisé Apache avec le mode Prefork, pour des raisons de performance, le paragraphe suivant donne plus d'éclaircissement.

II.2.2.3 Pourquoi utiliser Prefork ?

Le module mpm-worker est le plus pratique pour l'aspect "scalable" tandis que le "Prefork" a l'avantage d'être très stable. Or, La stabilité et la performance sont les points les plus importants pour un site web [LIN 08].

Prefork présente des avantages par rapport à worker dans certaines situations : il peut être utilisé avec les modules tiers qui ne supportent pas le threading (e.g. PHP). Si une erreur se produit avec une requête, en utilisant le Prefork, le problème est encapsulé à un processus et n'affecte pas les autres requêtes, qui peuvent ne pas être le cas à l'aide d'un serveur multi-thread.

II.2.2.4 Cycle de vie de traitement d'une requête

Le traitement d'une requête en Apache s'effectue en plusieurs points. Les modules d'Apache permettent de gérer un aspect spécifique du traitement d'une requète http : Contrôle d'accès (mod_access), authentification (*mod_auth*), interprétation du PHP (*mod_php*), etc.

La figure II.2 montre les différentes phases de traitement d'une requête [AND 05a]:

- **Traduction de l'URI en nom de fichier** *(Translate Name)* **:** Cette étape permet de convertir l'URI en nom de fichier ou plutôt en chemin interne où peut être trouvé le document sur le serveur.
- **Analyse de l'entête de la requête** *(Header parser)* **:** Le serveur analyse les entêtes HTTP de la requête, ce qui lui permet d'exécuter certains traitements en fonction de ces entêtes.
- **Contrôle d'accès** *(check Access) :* Des restrictions d'accès peuvent être définies sur les ressources du serveur, en fonction de certaines caractéristiques du client (en général son adresse IP. ou le nom de sa machine).
- **Vérification d'identité** *(check User ID) :* On vérifie si le mot de passe et le login fournis par le client existent et qu'ils sont valides
- **Vérification de l'autorisation d'accès** *(check Auth) :* Certaines ressources du serveur sont protégées, en plus de l'authentification par login et mot de passe, par certaines directives : accès uniquement si le client est propriétaire du fichier, ou seulement s'il appartient à un groupe d'utilisateurs donné. Cette étape vérifie que le client a bien le droit d'accéder au fichier demandé et si celui ci est protégé par les directives précédentes.
- **Détermination du type MIME de l'objet requis** *(Type checker) :* Il s'agit ici de déterminer quel est le type MIME du document demandé afin d'exécuter certaines actions (par exemple s'il s'agit d'un fichier CGI, il faudra le traiter en conséquence).
- **Emission de la réponse vers le client :** L'entête HTTP de la réponse est constituée et envoyée au client, puis c'est le document lui-même (ou le résultat d'un traitement le cas échéant) qui est envoyé au client.
- **Journalisation** *(Logger) :* On enregistre une trace de la transaction effectuée en enregistrant dans un ou plusieurs fichiers de logs (suivant la configuration d'Apache) certaines données de cette transaction.

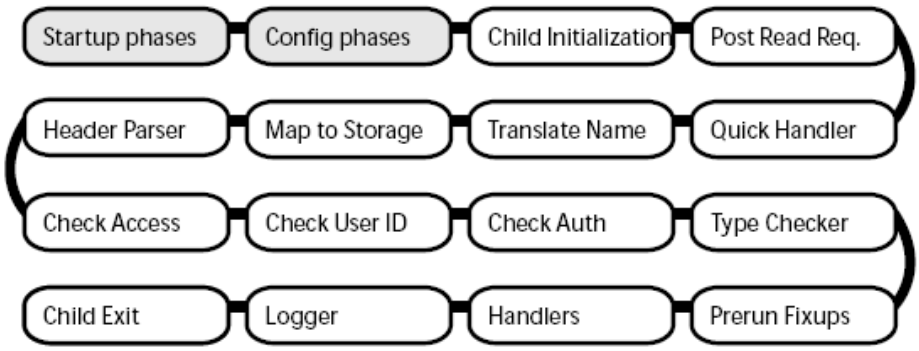


Figure II.2 : Cycle de vie de traitement d'une requête par Apache [AND 04]

II.2.3 Organisation du code source d'Apache

Nous avons utilisé la distribution 2.0.54 d'Apache http server. La figure II.3 montre comment le code est organisé.

La distribution d'Apache 2.0.45 comporte 2165 fichiers répartis dans 183 sous-répertoires ; 704 fichiers contiennent le code source en C (environs 280 000 lignes y compris les commentaires). Néanmoins, la partie importante pour nous est localisée dans peu de répertoires centraux. Les fichiers du noyau (core source) se localisent dans le répertoire *server*, les MPMs sont localisés dans le sous-répertoire *mpm*. Le répertoire *module* contient des sous-répertoires avec tous les modules inclus dans la distribution.

II.2.3.1 Modules

La structure modulaire d'Apache constitue l'une de ses principales forces. Le noyau principal de l'exécutable ne contient qu'un ensemble de fonctionnalités fondamentales, le reste étant apporté par le biais des modules, soit intégré de façon standard à Apache, soit chargé dynamiquement lors de son lancement (voir figure II.3).L'ajout d'un nouveau module se borne généralement à l'installer, puis à redémarrer le serveur Apache et rien de plus.

Il existe de très nombreux modules tiers pour Apache. Certains d'entre eux, comme *mod_fastcgi,* procurent des fonctionnalités nouvelles permettant d'augmenter fortement la puissance d'Apache. *mod_fastcgi* permet à apache de mettre en cache des scripts CGI de manière à les rendre à la fois plus efficaces pour les utilisateurs et moins exigeants en ressource système. D'autre modules procurent un gain appréciable tant en puissance qu'en flexibilité : *mod_perl* fournit à Apache un interpréteur Perl complet, lui donnant accès aux innombrables programmes Perl disponibles.

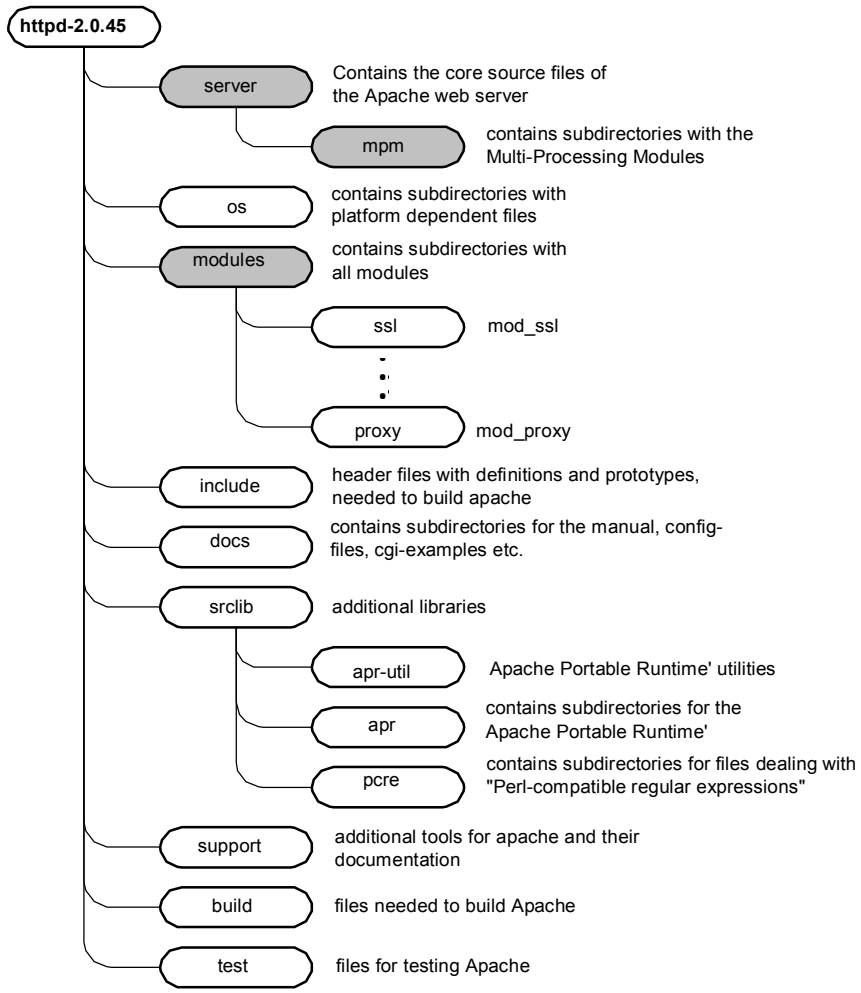


Figure II.3: Structure du code source d'Apache 2.0.45 [GRO 04]

II.2.4 Directives Apache liées à la performance [WAI 00]

Apache est configuré d'après ses directives. Il dispose de plusieurs directives permettant d'ajuster son niveau de performance en différentes occasions [WAI 04]. Toute modification apportée à une directive doit être ajoutée au fichier de configuration d'Apache (*httpd.conf*), afin quelle soit prise en charge lors du lancement du serveur Apache ou bien lors d'un *Graceful restart* (voir section II.2.6).

Elles se répartissent en deux groupes principaux :

- Directives de niveau processus, contrôlant le nombre de processus (ou threads, sous Windows) que Apache démarre et maintient en réserve pour traiter les requêtes entrantes.
- Directives de niveau protocole, contrôlant la façon dont Apache gère les connexions avec les clients et combien de temps il doit attendre une activité avant de mettre de lui-même fin à une connexion.

II.2.4.1 Directives de performance relatives aux processus

Apache s'exécute sur une plate-forme UNIX comme serveur à préduplication de processus. Cela signifie que, lors de son démarrage, il crée une réserve de processus fils prête à gérer les requêtes clients entrantes. Au cours du traitement des requêtes, Apache essaye de s'assurer qu'il reste au moins quelques serveurs libres capables de traiter d'autres requêtes, Apache dispose de trois directives contrôlant les jeux de processus.

- **StartServers <nombre>** (valeur par défaut 5)

Cette directive définit le nombre des processus enfants créés au démarrage d'Apache. Comme celui-ci contrôle toutefois dynamiquement le nombre de processus en fonction de l'activité du serveur, ce réglage n'a pas beaucoup d'effet, il est donc inutile de le modifier. Si par exemple *MinSpaceServers* lui est supérieur, Apache fait naître immédiatement de nouveaux processus fils.

- **MinSpareServers <nombre>** (valeur par défaut 5)

Cette directive définit le nombre minimal de processus devant être disponibles à tout instant. Si les processus sont occupés à traiter des requêtes client, Apache démarre de nouveau processus pour conserver ce nombre minimal de processus libres dans sa réserve. L'augmentation de cette valeur ne présente d'intérêt que dans le cas d'un site qui doit traiter simultanément un grand nombre d'appels.

- **MaxSpareServers <nombre>** (valeur par défaut 10)

Cette directive définit le nombre maximal de processus Apache qui peuvent être simultanément inactifs. Si de nombreux processus sont démarrés afin de gérer un pic de demandes, puis que ceux-ci diminuent, cette directive s'assure que les processus en surplus ne continuent pas à s'exécuter. Pour être pertinente, sa valeur doit être supérieure ou égale à *MinSpareServers*.

Ces directives étaient dans le passé bien plus importantes qu'elles ne le sont désormais. Depuis sa version 1.3, Apache possède un algorithme de gestion des requêtes entrantes extrêmement efficace, créant entre 1 et 32 nouveaux processus chaque seconde jusqu'à ce que toutes les requêtes clients soient satisfaites. L'objectif est d'éviter le démarrage simultané d'un nombre trop élevé de processus, à moins que cela ne soit absolument nécessaire. Le serveur débute avec un processus, puis double le nombre créé chaque seconde (jusqu'au maximum de 32). Donc, si Apache rencontre réellement une forte augmentation de la demande, il démarrera plusieurs processus.

La conséquence de cette stratégie est que la méthode de gestion dynamique de la réserve adoptée par Apache est parfaitement capable de s'adapter à de larges variations de la demande.

- **MaxClients <nombre>** (valeur par défaut 256)

Cette directive fixe la limite maximale des requêtes simultanées que le serveur peut prendre en charge ; aucun processus enfant au-delà de ce nombre n'est engendré. Il ne devrait pas être défini à une valeur trop basse, sinon un nombre toujours croissant de connexions est reportée dans la file d'attente et occasionne finalement un dépassement du temps imparti, alors que les ressources du serveur restent inutilisées. S'il est réglé à une valeur trop élevée, en revanche, le serveur commence à « swapper », ce qui fera diminuer considérablement le temps de réponse.

Cette directive est la plus effective pour augmenter les performances du serveur web Apache. S'il y a plus d'utilisateurs simultanés que *MaxClients*, les requêtes seront mises en file d'attente jusqu'à un nombre défini en fonction de la directive *ListenBacklog*.

- **MaxRequestsPerchild <nombre>** (valaur par défaut 0)

Cette directive fixe la limite maximale du nombre de requêtes pouvant être gérées par un processus Apache particulier avant de prendre fin volontairement. Elle est par défaut fixée à zéro, si bien qu'un processus ne se termine jamais de lui-même.

II.2.4.2 Directives de performance relatives aux protocoles

Outre le contrôle de la réserve du serveur, Apache dispose également des directives permettant de contrôler les performances au niveau des protocoles TCP/IP et HTTP. Comme ces protocoles sont indépendants des plates-formes, ces directives s'appliquent quelle que soit la plate forme sur laquelle s'exécute Apache.

- **KeepAlive <on/off>**

KeepAlive permet à un client d'effectuer plusieurs requêtes HTTP en séquence sur la même connexion, pourvu que le client ait indiqué qu'il en est capable et qu'il souhaite le faire. C'est particulièrement utile pour la prise en charge des pages web comportant de nombreuses images. Si *KeepAlive* est positionnée sur Off, alors une connexion TCP (voir section I.5) séparée doit être créée pour chacune des images. La surcharge causée par l'établissement des connexions TCP peut être éliminée en positionnant *KeepAlive* sur On.

KeepAlive permet un dialogue plus rapide entre un client et le serveur, au risque d'empêcher le processus concerné du serveur de pouvoir gérer toute autre requête tant que le client n'est pas déconnecté. Afin de prendre cela en compte, Apache propose deux autres directives afin de gérer la durée de vie d'une connexion permanente.

- **KeepAliveTimeout <seconde>**

Cette directive spécifie le temps accordé à un processus Apache (ou à un thread, sous Windows) dans l'attente d'une requête HTTP par le même client. Cela devrait être une valeur relativement faible, et la valeur par défaut est de 15 secondes, ce qui est équivaut à :

KeepAliveTimeout 15

Cette valeur doit être légèrement supérieure au temps maximal que le serveur devrait passer à générer et à envoyer une réponse.

- **MaxKeepAliveRequests <nombre>**

Indépendamment de la valeur attribuée à *Timeout*, une connexion permanente prendra également fin automatiquement dès que le nombre de requêtes fixé par *MaxKeepAliveRequests* est atteint. Afin de garantir le niveau de performance du serveur, cette valeur doit être élevée : la valeur par défaut est fixée à 100.

MaxKeepAliveRequests 100

Fixer cette valeur à zéro fait en sorte que les connexions permanentes restent actives pour toujours, tant que le délai de *Timeout* n'est pas dépassé et que le client ne se déconnecte pas.

- **TimeOut**

Cette directive détermine combien de temps une connexion HTTP sera autorisée par Apache alors qu'elle est apparemment inactive. Elle est définie selon les critères suivants :

- La durée écoulée depuis qu'une connexion a été établie et une requête *GET* reçue (voir section I.6.1), cela n'affecte pas les connexions permanentes, pour lesquelles *KeepAliveTimeout* joue ce rôle.
- La durée écoulée depuis la réception du dernier paquet de données selon une requête HTTP *post* ou *put* (voir section I.6.1).
- La durée écoulée depuis la réception de la dernière réponse ACK (*acknowledgement voirsection* I.5.1) si le serveur attend d'autres.

La valeur par défaut de TimeOut est de 5 minutes, ce qui est équivaut à : TimeOut 300

- **ListenBacklog**

Les requêtes de connexion émanant des clients s'empilent dans une file d'attente (Backlog file) jusqu'à ce qu'un processus Apache soit libre afin de les servir (figure II.4). La longueur maximale de la file d'attente est définie par la directive *ListenBacklog,* dont la valeur par défaut est 511.

ListenBacklog 511.

511 connexions TCP seront en attente jusqu'à ce qu'un processus Apache soit libre. Beaucoup de systèmes d'exploitation réduisent cette valeur à celle d'une limite système.

II.2.5 La File d'attente Backlog

Toute application appelant la fonction listen() lui passe 2 paramètres, l'un étant la taille de la file d'attente, la backlog. Apache n'obéit pas à la règle et utilise une backlog DEFAULT_LISTENBACKLOG dont la longueur est fixée par défaut à 511, cette dernière peut être modifiée grâce à la directive *ListenBacklog*.

Linux réduit la taille de la backlog à la valeur de *somaxconn* si elle lui est supérieure [LIN 10a]. Donc, lors de son lancement, Apache n'a plus une backlog de 511 mais uniquement de 128, valeur par défaut de *somaxconn*. Sous Linux, *somaxconn* détermine la taille maximale de la file d'attente pour les sockets totalement établies en attente d'acceptation. En revanche, si une nouvelle connexion arrive, alors, que la file est pleine, le client reçoit une erreur indiquant « SERVICE UNAVAILABLE » (voir Annexe B).

Exemple : Considérons un système qui utilise le serveur web Apache avec : MaxClients réglé à 100 et utilise une connexion permanente (KeepAlive). Si tous les 100 processus fils sont occupés alors qu'une nouvelle demande est envoyée au serveur, la connexion est toujours établit mais elle ne peut pas être acceptée immédiatement par Apache, elle est mise en attente dans la file « backlog queue » (figure II.4). Dès que l'une des 100 connexions permanentes dépasse le délai de *Timeout,* la nouvelle demande sera acceptée et exécutée. La discipline de service est FIFO au sein du *backlog*.

Le serveur web Apache utilise deux types de sockets :

Listen socket : Utilisée pour établir une connexion avec le client. Ce type peut être aussi désigné sous le nom "*dialup socket*" ou "*server socket*".

Connection socket : Le serveur emploie ce type de socket pour l'échange des données avec le client.

La figure II.4 montre un exemple illustrant l'utilisation de ces deux types de sockets par le serveur Apache pour communiquer avec le navigateur du client. Du côté serveur, dans cet exemple, il y a juste une socket d'écoute connectée au port TCP du serveur (habituellement le port 80). Seulement un seul processus fils (Listner) est autorisé à utiliser la socket d'écoute (*Listen socket*) à la fois. Le navigateur du client utilise la socket *Connection-socket* pour établir une connexion avec la socket *Listen-socket*. Si tous les processus fils sont occupés, alors toute nouvelle demande d'un client est placée dans la file d'attente *backlog-queue.*

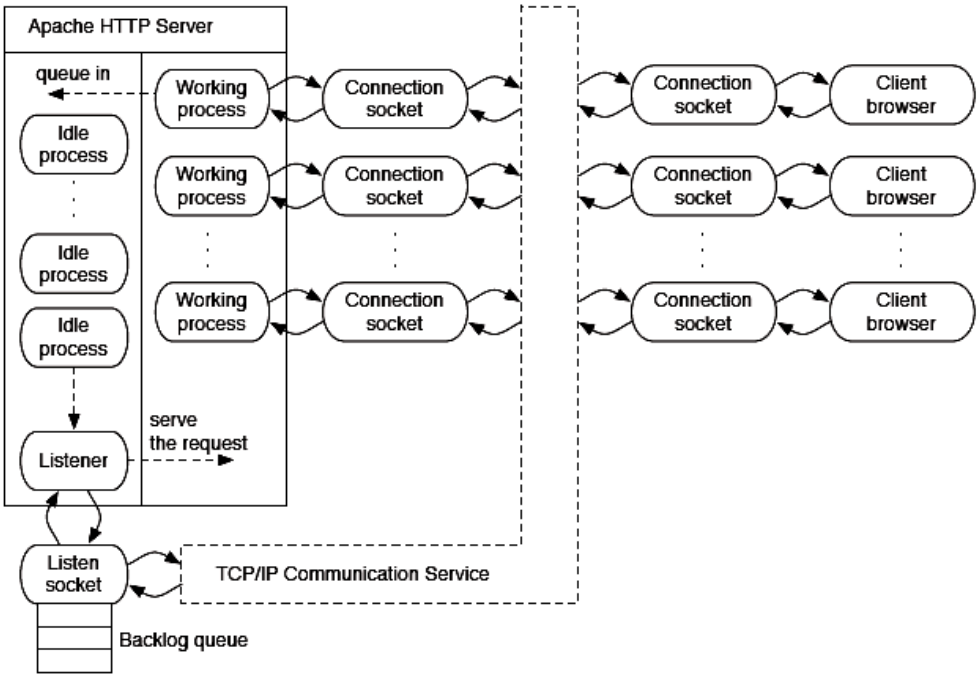


Figure II.4 : Interaction Client/ serveur web Apache [LIN 08]

Le processus fils connecté à *Listen-socket* utilise un appel système nommé *accept* pour accepter la demande qui était en attente dans la backlog. L'appel système *accept* va créer et retourner une *Connection-socket* vers le client, celle-ci est utilisée par le processus fils pour communiquer avec le client. Le processus fils libérera alors la *Listen-socket* de sorte qu'un autre processus fils libre puisse l'utiliser pour attendre la prochaine demande entrante et commence à traiter la demande du client via la socket *Connection-socket, dès* qu'il termine (et quand le *Timeout* expire) il revient à l'état attente avec les processus libres.

II.2.6 Graceful restart

Les processus enfants (child process) obtiennent les informations de configuration lors de leurs créations par le processus père (Master process). Cependant, si l'Administrateur veut appliquer des changements de configuration au serveur, il doit effectuer un redémarrage *graceful restart,* qui permet à des processus enfants de compléter le traitement des requêtes en cours avant d'être remplacés par des nouveaux processus de la nouvelle configuration. De cette manière on préserve la disponibilité du service.

II.3 La modélisation du serveur web Apache par file d'attente

II.3.1 La théorie des files d'attente

Les origines de la théorie des files d'attente remontent à 1909 à l'époque où A. K. Erlang en a posé les bases dans ses recherches sur le trafic téléphonique. Aujourd'hui, la théorie des files d'attente fait partie intégrante de l'évaluation des performances des systèmes informatiques La théorie des files d'attente est un formalisme mathématique qui permet de mener des analyses quantitatives à partir de la donnée des caractéristiques du flux d'arrivées et des temps de service [KOB 78].

Définition. Une file d'attente simple est composée d'un buffer et d'un ou plusieurs serveurs. Le processus d'attente, ou la constitution de la file commence à se manifester dès que le taux des arrivées excède le taux de service (par taux, on entend le nombre moyen de clients arrivants ou servis par unité de temps).

Le processus décrit est stochastique, car on ne peut pas connaitre à l'avance ni le temps d'arrivée d'un client, ni la durée de service qu'il demandera. On notera λ, le nombre moyen d'arrivées d'un client par unité de temps (le taux d'arrivée) et μ le nombre moyen de clients par unité de temps que le serveur peut servir (le taux de service). Le temps de service moyen est donc $1/\mu$.

Un système de file d'attente pourrait être représenté par la figure II.5 :

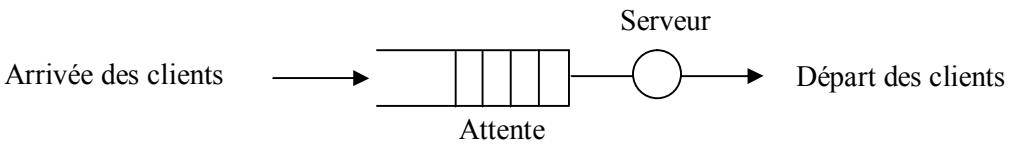


Figure II.5 : Système à un seul serveur

II.3.1.1 Les concepts de base des files d'attente

a. Processus d'arrivée

Les clients arrivent au sein du système en décrivant un processus déterminé. Le modèle le plus simple et le plus courant est celui des arrivées complètement aléatoires, ce qui est caractérisé par *le processus de poisson*, i.e. les temps inter-arrivées suivent une loi exponentielle de paramètre λ, et les temps de service des clients suivent une loi exponentielle de paramètre μ.

b. Processus de service

La deuxième composante d'un système de files d'attente est la quantité de service demandée par un client ; chaque client qui entre dans la file d'attente transporte avec lui sa demande de service.

c. Discipline de service

Une fois le serveur devient libre, il choisit un client de la file suivant une politique ou une discipline adoptée au sein du système. Les disciplines de service les plus courantes sont :

- **FIFO** : First In First Out, service dans l'ordre des arrivées ;
- **LIFO** : Last In First Out, service inverse de l'ordre des arrivées ;
- **PS** : Processor Sharing, serveur partagé entre tous les clients présents ;
- **RSS** : Random Selection for Service, sélection aléatoire de service ;
- **Priorité** : plusieurs classes de clients de priorité différente ; le client ayant la plus haute priorité est servi en premier, et la discipline de service est FIFO au sein d'une même classe.

II.3.1.2 Théorème de Little

Considérons un système d'attente quelconque. Soit λ le taux d'arrivée des clients dans le système, N le nombre moyen des clients dans le système, et T le temps d'attente moyen des clients dans le système. Alors $N = \lambda T$.

Cette loi est très utilisée dans la pratique pour dimensionner un système ou analyser rapidement sa performance.

II.3.1.3 La loi exponentielle

Soit $\lambda>0$. On appelle loi exponentielle de paramètre λ, la loi de probabilité sur R^+ de densité $\lambda e^{-\lambda x}$ $x \geq 0$. Si le temps *t* entre deux arrivées suit la loi exponentielle de paramètre θ, le nombre d'arrivées dans un intervalle de temps [0, t] suit la *loi de poisson* de paramètre $\lambda = \theta t$:

$$P[N=n] = \frac{e^{-\theta t}(\theta t)^n}{n!}$$

II.3.1.4 La notation de Kendall

Pour mieux s'y retrouver dans les nombreuses variantes possibles dans la manière dont fonctionne une file d'attente, on utilise la notation de Kendall. C'est un symbole de la forme

A/ S/ P/ K/ D

- **A** désigne la loi des inter-arrivées (durée entre deux arrivées successives) ; Selon la nature de cette loi, **A** peut prendre la valeur :
  - M pour la distribution exponentielle (M comme Markov) ;
  - G pour une distribution quelconque, G = générale ;
  - U pour une distribution uniforme.

  Cette liste n'est pas exhaustive.

- **S** désigne la loi des durées de service des clients ; Classification identique à celle des inter-arrivées ;
- **P** désigne le nombre de serveurs. **P** peut prendre n'importe quelle valeur entière, y compris l'infini. Par défaut, c'est 1.
- **K** désigne la capacité de la salle d'attente, hors serveur.
- **D** désigne la discipline de service, ou encore : politique de service, précisant comment les clients sont servis. D peut prendre les 'valeurs' suivantes : FIFO, LIFO, PS (Processor Sharing), priorités. Par défaut, D=FIFO.

II.3.1.5 Chaînes de Markov en temps discret

La théorie des processus stochastiques (en particulier les chaînes de Markov) permet de calculer les probabilités d'état stationnaires.

Un processus stochastique en temps discret et à espace discret {X(n), n ∈ N}, à valeurs dans un espace d'état $\mathcal{E}$, est une chaîne de Markov si et seulement si les deux propriétés suivantes sont vérifiées :

- pour tout n ∈ N , et tout vecteur ($j_0, j_1, \ldots\ldots, j_n, j_{n+1}$) d'éléments de $\mathcal{E}$
  $P[X(n+1)=j_{n+1}/ X(n) = j_n, \ldots, X(0)= j_0] = P[X(n+1) = j_{n+1} / X(n) = j_n]$.

- pour tout n ∈ N, et toute paire (i, j) de $\mathcal{E}$
  $P [X(n+1) = j / X(n)= i)] = P(i, j)$, indépendamment de t.

En d'autres termes, une chaîne de Markov en temps discret a la propriété que son évolution (passage de *X(n) à X(n+1)*) ne dépend que de l'état courant *X(n)* et non pas de son passé (les états visités aux instants 0,1,……., n-1).

Les quantités *P(i, j)* sont *les probabilités de transition de la chaîne*. Ainsi, *P(i, j)* est la probabilité que la chaine de Markov passe de *i* à *j* en une étape (ou en une unité de temps).

II.3.1.6 Chaînes de Markov en temps continu

Dans le cas continu le processus stochastique est observé à des instants arbitraires. Un processus $\{X_t, t \in R^+\}$ est une chaîne de Markov en temps continu, aussi appelé processus de Markov, homogène si et seulement si :

- pour tout n ∈ N $0 < t_1 < t_2 < \ldots. < t_n < t_{n+1}$, et vecteur ($j_0, j_1, \ldots\ldots, j_n, j_{n+1}$) d'éléments de $\mathcal{E}$:
  $P [X(t_{n+1})=j_{n+1}/ X(t_n) = j_n, \ldots, X(t_0)= j_0] = P[X(t_{n+1}) = j_{n+1} / X(t_n) = j_n]$.

- pour tout réels s, t et u, et toute paire (i, j) de $\mathcal{E}$
  $P [X(t+u)=j / X(s+u) = i] = P [X(t) = j / X(s) =i] = P_{t-s}(i, j)$. Indépendamment de t.

De même que dans le cas du temps discret, on s'intéresse aux probabilités d'état :

$P_t(n) = P[X(t) = n]$, $t \in R^+$, $n \in \mathcal{E}$. La probabilité d'avoir *n* clients à l'instant *t*

L'état d'un système à l'instant *t* est le nombre (*n)* de clients présents dans le système à cet instant.

II.3.1.7 Files markoviennes

La théorie des files d'attente s'appuie sur l'analyse markovienne pour étudier les files d'attente. Ces méthodes markoviennes permettent d'étudier et d'exprimer de façon analytique le temps d'attente moyen dans certains systèmes d'attente classiques.

Le processus d'arrivée des clients est un processus de Poisson, La politique de service est de type FIFO.

II.3.1.8 La file M/M/1

La file M/M/1 est une file d'attente à un serveur, de capacité infinie, et de discipline de service FIFO. Les arrivées se font selon un processus de Poisson homogène de paramètre **λ** et les durées de service sont identiquement distribuées selon une loi exponentielle de paramètre **μ**.

Soit X(t) le nombre de clients dans le système, donc en attente et en service, à l'instant t. La propriété de la file M/M/1 est que le processus $\{X(t), t \geq 0\}$ est la forme la plus simple qu'il soit d'une chaîne de Markov à temps continu.

Le nombre moyen de clients et le délai moyen dans une file M/M/1 sont respectivement :

$$N = \frac{\lambda}{\mu - \lambda} = \frac{\rho}{1-\rho} \qquad \text{et} \qquad T = \frac{1}{\mu - \lambda}$$ (avec $\lambda < \mu$ **:** la file est stable**)**

La quantité $\rho = \frac{\lambda}{\mu}$ est *le taux d'occupation du serveur* de la file M/M/1, i.e. la probabilité que le serveur soit occupé.

II.3.1.9 La file M/M/1/k

La file **M/M/1/k** possède un buffer de taille *k* : si un client arrive alors que la file d'attente est pleine (il y a déjà *k* clients dans le système), le client est rejeté.

La probabilité de rejet P_r ou taux de perte est :

$$P_r = \frac{(1-\rho)\rho^k}{1-\rho^{k+1}} \text{ si } \rho \neq 1, \text{ et } P_r = \frac{1}{k+1} \text{ si } \rho = 1$$

L'expression du nombre de clients dans une file M/M/1/k est

$$N = \frac{\rho}{1-\rho} - \frac{(k+1)\rho^{k+1}}{1-\rho^{k+1}} \text{ si } \rho \neq 1, \text{ et } N = \frac{k}{2} \text{ si } \rho = 1.$$

II.3.2 La modélisation du serveur par la file M/G/1/K*PS

La plus part des serveurs web utilisent, en pratique, une politique de file d'attente (les requêtes arrivent dans une seule file d'attente à un seul serveur web), en employant une stratégie d'ordonnancement de type FIFO (premier arrivé-premier servie) figure II.6.

Cao et al [CAO 03] ont proposé un modèle de file d'attente M/G/1/K*PS. Ce modèle représente une file d'attente à un serveur ; la loi des inter-arrivées est exponentielle, celle du service est générale. La taille de la salle d'attente est de K et la discipline de service est PS (Processor Sharing) : les clients en attente reçoivent la même portion de service, et ce, à tour de rôle. L'arrivée d'un nouveau client est rejetée si la capacité maximale de K clients est atteinte. Quand un client reçoit le temps moyen de service requis, il quitte la file. La file est représentée sur le schéma suivant :

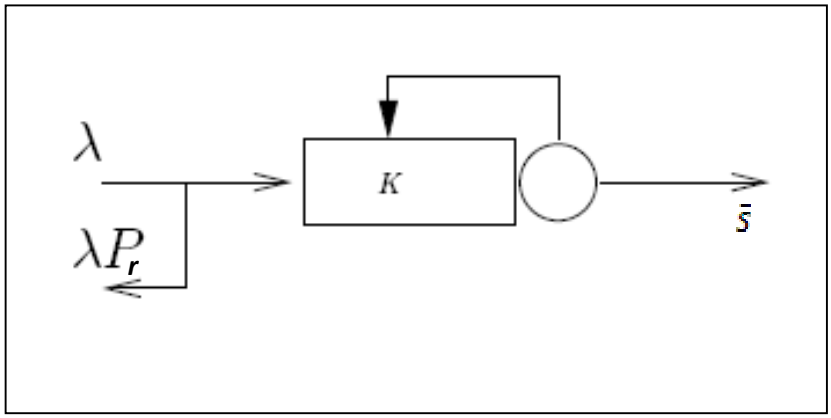


Figure II.6 : File d'attente avec une discipline de service PS [CAO 03].

La probabilité du nombre de clients dans le système à l'équilibre stationnaire est donnée par la formule :

$$P[N= n]= \quad \frac{(1-\rho)\rho^n}{1-\rho^{k+1}} \qquad (2.1)$$

Où **ρ** représente le taux d'occupation du système. Ce dernier est égal au trafic d'arrivée **λ** multiplié par le temps moyen de service $\overline{S}$**.**

Le serveur web est modélisé par la file d'attente M/G/1/K*PS représentée sur la figure II.6 Le trafic d'arrivée est un processus de Poisson de paramètre **λ**. Le temps moyen de service requis pour chaque requête est $\overline{S}$. Le serveur ne peut pas traiter plus de **K** requêtes à la fois. L'arrivée d'une requête est rejetée si le nombre de clients dans la file est égal à **K**. La probabilité de rejet est notée Pr. Ainsi, le nombre de connexions rejetées est égal à **λ Pr.**

Les expressions des trois métriques de performance : *Le temps moyen de réponse, le débit et la probabilité de rejet*, sont déduites à partir de la formule (2.1).

La probabilité de rejet **P_r** est égale à la probabilité d'observer **K** clients dans le système à l'équilibre stationnaire :

$$P_r = P[N = K] = \frac{(1-\rho)\rho^n}{1-\rho^{k+1}} \quad (2.2)$$

Le débit **D** est la vitesse de connexion des requêtes satisfaites. Quand le serveur est dans un état stable, **D** est alors égal à la vitesse de connexion des requêtes acceptées.

$$D = \lambda (1-P_r) \quad (2.3)$$

Le temps moyen de réponse **R** est obtenu à partir de la loi de Little définie par :

$$R = \frac{E[N]}{D} = \frac{\rho^{k+1}(K\rho - K - 1) + \rho}{\lambda(1-\rho^k)(1-\rho)} \quad (2.4)$$

- **Estimation des paramètres**

Dans ce modèle, deux paramètres sont à estimer : le temps moyen de service $\overline{S}$ et la capacité de la file **K**. en effet, Cao et al [CAO 03] ont estimé ces derniers par maximisation de la fonction *vraisemblance* du temps moyen de $\overline{S}$ réponse observé. Le temps de réponse d'une requête HTTP est ensuite estimé comme le temps de réponse moyen de traitement d'un client dans la file pour un taux d'arrivée. L'estimation des paramètres devient un problème d'optimisation :

$$(\hat{\overline{S}}, K) = \underset{(\overline{s},K)}{\arg\min} \sum_{i=1}^{m} \frac{(\hat{R}_i - R_i)^2}{\sigma_i^2 / n_i} \quad (2.5)$$

Les valeurs optimales des paramètres ont été obtenues par une technique de descente de gradient classique.

Où : **R_i** le temps moyen de réponse obtenu à partir du modèle et $\hat{R}_i$ le temps moyen de réponse estimé à partir de la mesure pour un trafic observé en entrée de débit λ_i, i=1…m.

La variance σ_i^2 est approximée par la variance estimée du temps de réponse $\hat{\sigma}_i^2$.

II.4 Conclusion

Dans ce chapitre, nous avons présenté en premier temps, le fonctionnement interne du serveur web Apache. Nous avons mis en évidence les directives d'Apache liées à la performance. Les plus effectives parmi celles-ci sont Maxclient, Keepalive et *ListenBacklog*.

Dans un second temps, nous avons présenté la modélisation du serveur Apache par la file d'attente M/G/1/K*PS. Cela nous a permis de choisir les critères de performance du serveur web qui représenteront la sortie de notre modèle (Cf. chapitre 4).

Chapitre III

Réseaux de neurones

III.1 Introduction

Les réseaux de neurones artificiels ont la spécificité d'imiter le fonctionnement du cerveau humain, qui se compose d'un nombre énorme de neurones. Ces derniers sont organisés dans un immense réseau, qui apparaît simple mais dont le fonctionnement est extrêmement compliqué. Le début de recherches sur les réseaux de neurones artificiels remonte aux années 1940 par les travaux de W.S.McCulloch et W.Pitts.

Les réseaux de neurones constituent maintenant une technique de traitement de données bien comprise et maitrisée, constatée par leurs applications dans plusieurs domaines à savoir : l'aéronautique, l'étude spatiale, la reconnaissance de formes, la robotique, le filtrage d'informations textuelles, la classification, la régression, etc.

Toutefois, ils n'ont pas encore atteint leur plein développement, pour des raisons plus psychologiques que techniques, liées aux connotations biologiques du terme, et au fait qu'ils sont considérés, à tort, comme des outils d'Intelligence Artificielle [DRE 02].

Dans ce chapitre, nous présenterons la technique des réseaux de neurones formels d'une façon qu'elle doit être considérée comme une extension puissante de techniques mathématiques, telles que la régression.

Nous nous concentrons essentiellement sur l'utilisation de réseaux de neurones *feedforward* qui forment une classe de modèles paramétriques performante pour la régression non-linéaire.

III.2 Pourquoi les réseaux de neurones ?

On a recours au réseau de neurones lorsqu'on désire établir un modèle mathématique d'une dépendance entre les variables, et qu'on ne dispose que d'un ensemble de variables mesurées et d'un ensemble de mesures (relative à un processus de nature quelconque). En d'autres termes, on cherche un modèle mathématique qui relie des causes (entrées du modèle) à des effets (sorties du modèle).

On désigne par modèle « une boite noire » qui constitue la forme la plus primitive de modèle mathématique : il est réalisé uniquement à partir de données expérimentales ou observations ; il peut avoir une valeur prédictive, dans un certain domaine de validité, mais il n'a aucune valeur explicative.

Ainsi, les réseaux de neurones construisent des prédicteurs à partir d'exemples. On cherche à estimer la fonction génératrice des exemples ; en d'autres termes, une fonction de régression. Cependant, le réseau de neurones MLP est le réseau le plus puissant et le plus utilisé pour la régression non linéaire [SAM 06].

III.3 Généralités

III.3.1 Neurone biologique

Nous présentons le modèle biologique juste pour comprendre l'inspiration biologique qui à été à l'origine de la première vague d'intérêt pour les neurones formels, dans les années 1940 à 1970 [MCC 43, MIN 69].

Un neurone biologique (figure III.1) est une cellule spécialisée dans le traitement et la transmission de l'information, il est constitué de trois parties : l'axone, les Dendrites et le noyau. Les dendrites forment un maillage de récepteurs nerveux qui permettent d'acheminer vers le noyau du neurone des signaux électriques en provenance d'autres neurones. Celui-ci agit comme un intégrateur en accumulant des charges électriques. Lorsque la charge accumulée dépasse un certain seuil, par un processus électrochimique, il engendre un potentiel électrique qui se propage à travers son axone vers un autre neurone. Les contacts entre deux neurones (entre axone et dendrite) se font par l'intermédiaire des synapses

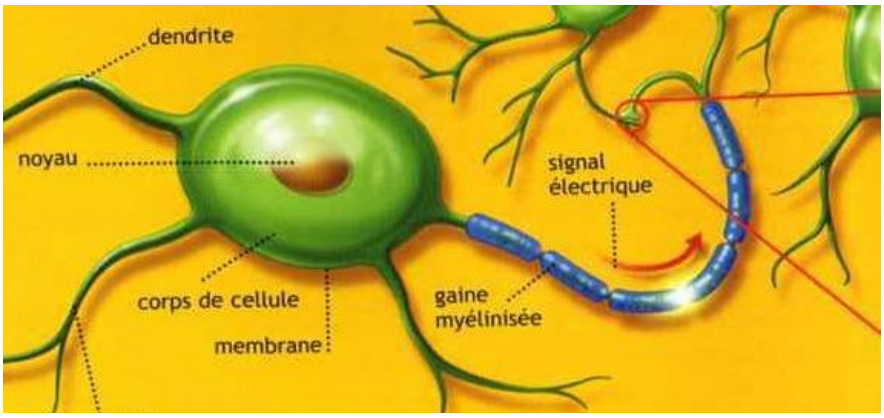


Figure III.1 : Neurone biologique.

III.3.2 Neurone formel

Le neurone formel (modèle mathématique) est illustré à la figure III.2. Un neurone est constitué d'un intégrateur qui effectue une combinaison, généralement la somme, des entrées $\{x_i\}$ pondérées par les paramètres $\{w_i\}$, qui sont souvent désigner sous le nom de

« poids » (*w*eight), ou, en raison de l'inspiration biologique des réseaux de neurones, « poids synaptiques ». La combinaison linéaire est appelée « potentiel » à laquelle s'ajoute un terme constant ou « biais » $\mathbf{w_0}$, appelé souvent le seuil d'activation du neurone :

$$\mathbf{y} = \mathbf{w_0} + \sum_{i=1}^{n} w_i x_i$$

Le résultat **y** est ensuite transformé par une fonction d'activation f qui produit la sortie du neurone.

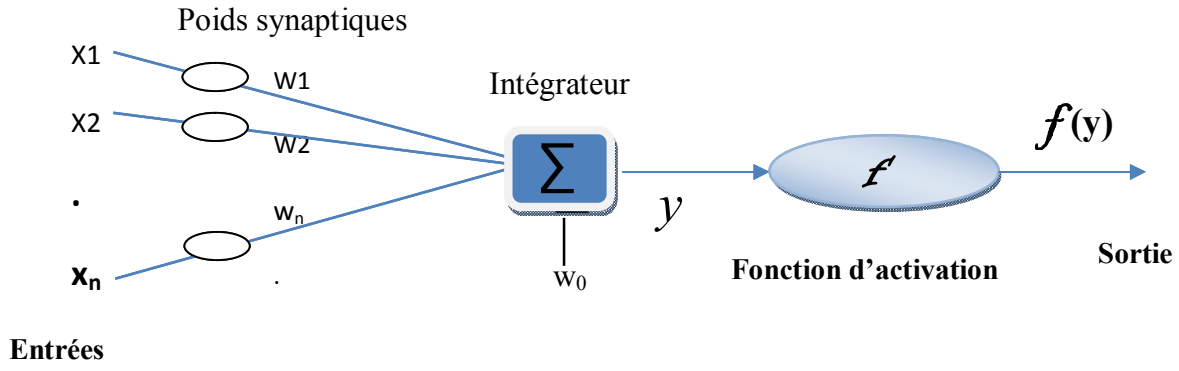


Figure III.2 : Neurone formel

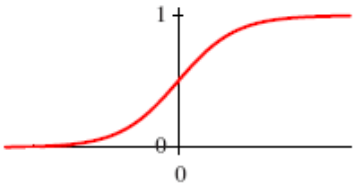
III.3.3 Fonction d'activation

La fonction d'activation joue un rôle prépondérant dans le comportement du neurone et du réseau entier. Plusieurs fonctions d'activation ont été examinées dans [DUC 99], la plus utilisé est la fonction *sigmoïde* (généralement c'est la fonction tangente hyperbolique) ; car elle introduit de la non-linéarité, et c'est aussi une fonction continue, différentiable.

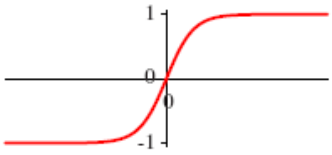
Les principales fonctions d'activation que l'on retrouve dans les différentes architectures sont représentées sur la figure III.3.

III.3.4 Définition d'un réseau de neurones

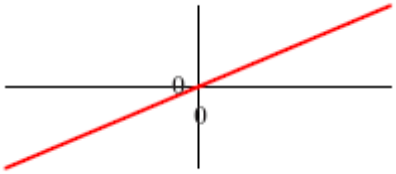
Un neurone (neurone formel) est une fonction algébrique non linéaire, paramétrée, à valeurs bornées [DRE 02] .Un neurone formel n'a d'intérêt que s'il est connecté à d'autres neurones pour former un réseau. Un tel réseau est appelé « Réseau de neurones ».



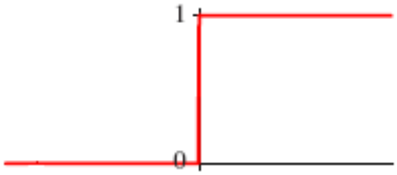
a) $f(x) = \dfrac{1}{1+e^{-x}}$



b) $f(x) = \tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



c) $f(x) = x$



d) $f(x) = \begin{cases} 1 & \text{Si x >0} \\ 0 & \text{Si non} \end{cases}$

Figure III.3 : Les principales fonctions d'activation : (a) fonction logistique, (b) fonction tangente hyperbolique, (c) fonction linéaire, (d) fonction à seuil

III.4 Réseaux de neurones pour la régression non-linéaire

III.4.1 La régression non linéaire

La régression non linéaire est une des méthodes classiques de la statistique largement utilisée dans le traitement des données. Son but principal est d'aider à la détermination d'une fonction univoque permettant de relier deux variables distinctes pour lesquels on fait l'hypothèse qu'il existe une relation de dépendance fonctionnelle. On se référera au livre de Bates [BAT 88] pour plus de détails.

Considérons une grandeur mesurable y_p, dépendant d'un ensemble de facteurs mesurés qui constituent les composantes d'un vecteur x. On considère les résultats des mesures y_p comme des réalisations d'une variable aléatoire Y, et les valeurs des grandeurs d'entrée comme des réalisations d'un vecteur aléatoire X. On cherche donc à estimer

l'espérance mathématique de la variable aléatoire Y pour une réalisation donnée x du vecteur aléatoire X ; on la note $\mathbf{E_Y(x)}$ Cette quantité est donc une fonction de *x* ; elle est appelée fonction de régression de la variable aléatoire Y.

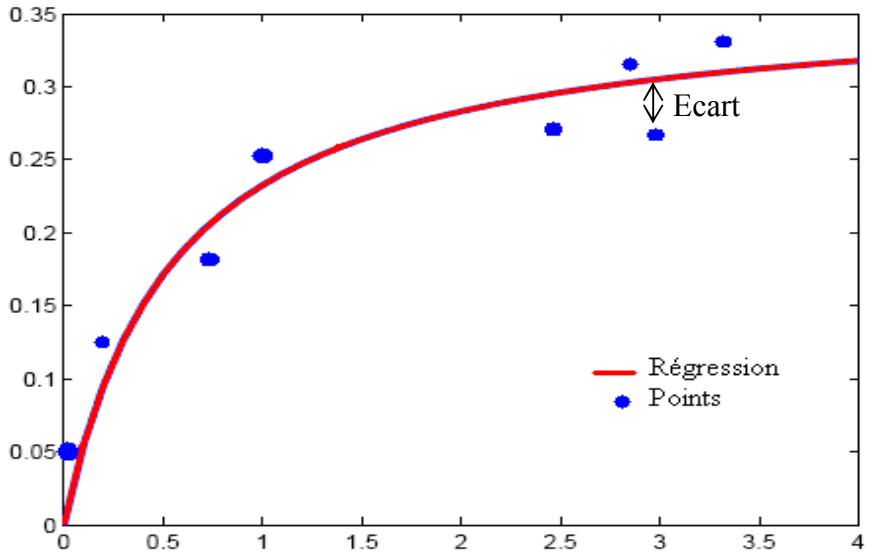


Figure III.4 : Courbe de régression non linéaire pour des données

III.4.2 Modèle non linéaire par rapport aux variables

Soit g(x,w) un modèle dont le vecteur des variable est x et le vecteur des paramètres est w. s'il existe un vecteur de paramètres w_p tel que le modèle soit identique à la fonction de régression $\mathbf{E_Y(x)}$ **:** $g(x,w_p) \equiv \mathbf{E_Y(x)}$**,** on dit que la famille des fonctions g(x,w) contient la régression, ou encore le modèle g(x,w) est vrai. Si ce n'est pas le cas, on se contentera de chercher un modèle aussi proche que possible de la fonction de régression $\mathbf{E_Y(x)}$**.** Or, un réseau de neurone est un modèle non linéaire par rapport aux variables, et la recherche d'un modèle proche de la fonction $\mathbf{E_Y(x)}$ n'est que l'apprentissage. Nous allons détailler ces points dans la section III.8

III.5 Architecture des réseaux de neurones

Un réseau de neurones est représenté par un graphe dont les nœuds sont les neurones et les arrêtes les connexions entre ceux-ci. Pour décrire l'architecture de réseau on emploi la matrice de poids de connexions $W = [\mathbf{w_{ij}}]$, où le wij dénote le poids de connexion du nœud j (ou de l'entrée j) au nœud i. quand wij=0, il n'y a pas de connexion entre le nœud j et le nœud i.

On distingue deux types de réseaux de neurones : les réseaux *non bouclés* et les réseaux *bouclés.*

III.5.1 Les réseaux de neurones non bouclés (réseau statique)

Sont des réseaux à propagation vers l'avant (*FNNs*, *feedforward neural networks)* [DU 06].Le réseau est organisé en couche successives, l'information est ainsi transmise de manière unidirectionnelle du neurone j vers le neurone i. la connexion entre deux couches successives peut être totale ou partielle (figure III.5).

Les données du problème (les entrées) sont présentées à la première couche cachée, la dernière couche est la couche de sortie représentant le résultat. Ainsi, les sorties peuvent être exprimées comme des fonctions déterministes des valeurs d'entrées.

Il n'existe aucune méthode optimale pour déterminer a priori la structure interne d'un réseau de neurones. Elle ne peut être déterminée que de façon expérimentale.

Les réseaux de neurones non bouclés sont souvent appelés *réseau statique* et cela au fait que : le temps ne joue aucun rôle fonctionnel dans un réseau de neurones non bouclé : si les entrées sont constantes, les sorties le sont également. Le temps nécessaire pour le calcul de fonction réalisée par chaque neurone est négligeable, et fonctionnellement, on peut considérer ce calcul comme instantané.

III.5.2 Les réseaux de neurones bouclés (réseau dynamique)

Le deuxième type de réseaux de neurones est les réseaux de neurones bouclés ou récurrents (*RNNs, Recurrent Neural Networks),* dont l'architecture est plus générale, le graphe des connexions est *cyclique :* lorsqu'on se déplace dans le réseau en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ. La fonction de neurone peut être d'elle-même ; pour cela, la notion de *temps* est explicitement prise en considération.
Ainsi, à chaque connexion d'un réseau de neurones bouclé est attaché outre que le poids, un *retard,* multiple entier de l'unité de temps choisis.

Un réseau de neurones bouclé à temps discret réalise une (ou plusieurs) *équations aux différences non linéaires*, par composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions [DRE 02].
La forme la plus générale des équations régissant un réseau de neurones bouclé est appelée forme canonique [NER 93] :

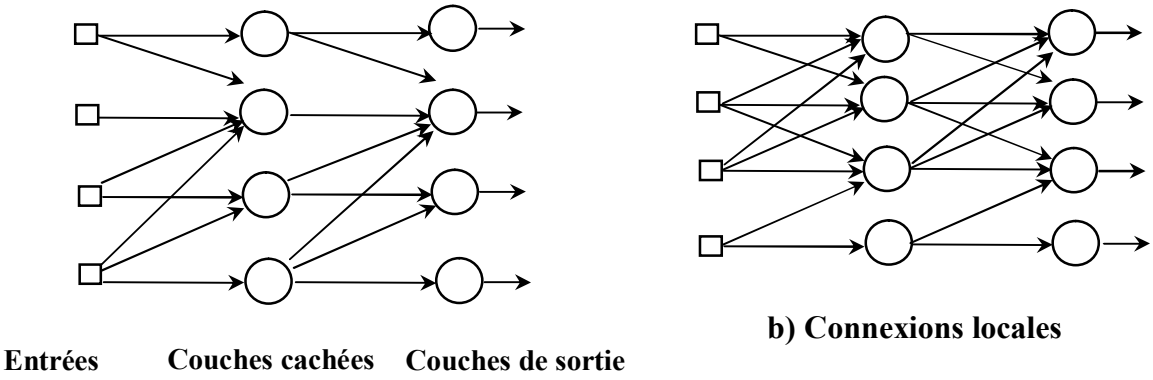
$$x(k+1) = \varphi[x(k), u(k)]$$

$$y(k) = \psi[x(k), u(k)]$$

Où φ et ψ sont des fonctions non linéaires réalisées par un réseau de neurones non bouclé, k désigne le temps (discret).

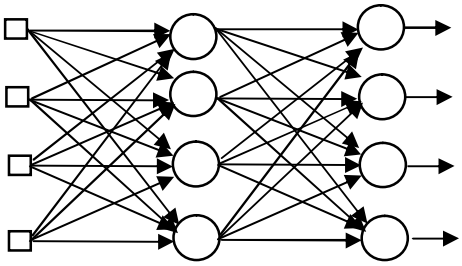
Le modèle de Hopfield [HOP 82] et la machine de Boltzman [ACK 85] sont les réseaux bouclés les plus populaire.

On peut se référer au livre de *K-L.Du* [DU 06] pour une présentation plus complète des réseaux de neurones bouclés.



a) Connexions directes

b) Connexions locales



c) Connexions totales

Figure III.5 Réseaux de neurones feedforward [PAL 06].

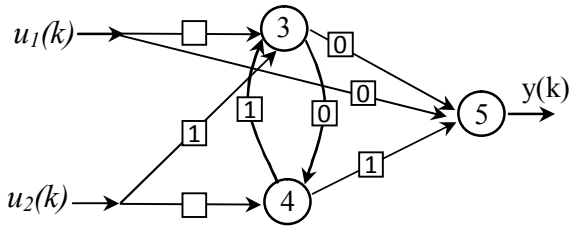


Figure III.6 : Exemple d'un réseau de neurones bouclé. Les nombres dans les carrés sont les retards (exprimés en nombre de périodes d'échantillonnage) associés à chaque connexion [DRE 02].

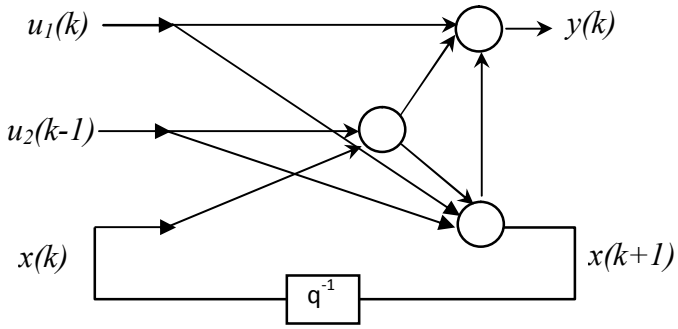
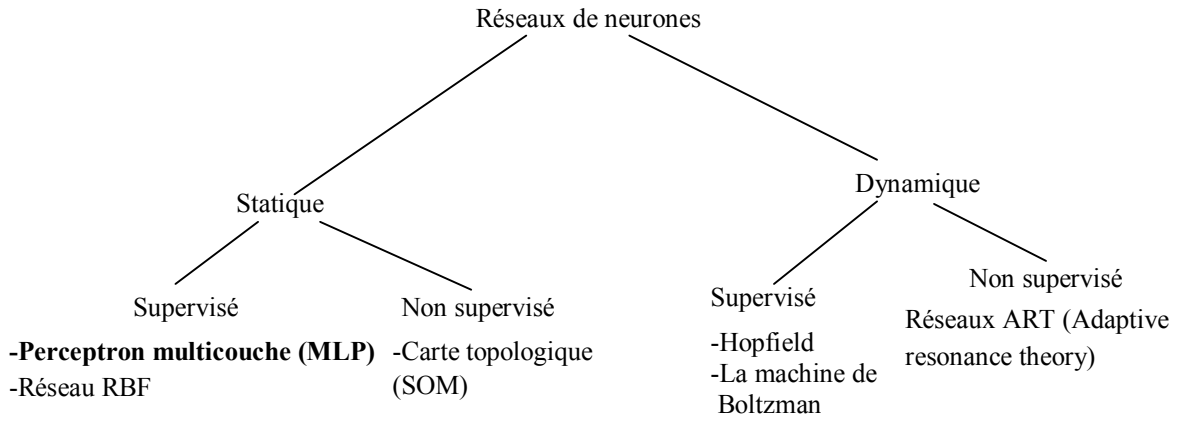


Figure III.7 : La forme canonique du réseau de la Figure III.6 [DRE 02].

III.5.3 Taxonomie des réseaux de neurones (non exhaustive)



III.6 Le perceptron multicouche MLP

Le perceptron multicouche (MLP, Multi Layer Perceptron) est un réseau de type FNNs (voir section III.5.1). Le réseau est organisé en couche successives. Un neurone ne peut être connecté qu'à la totalité des neurones d'une couche limitrophe. Ainsi il n'y a pas de connexions entre les neurones d'une même couche, et l'information circule des entrées vers les sorties sans retour en arrière (figure III.8).

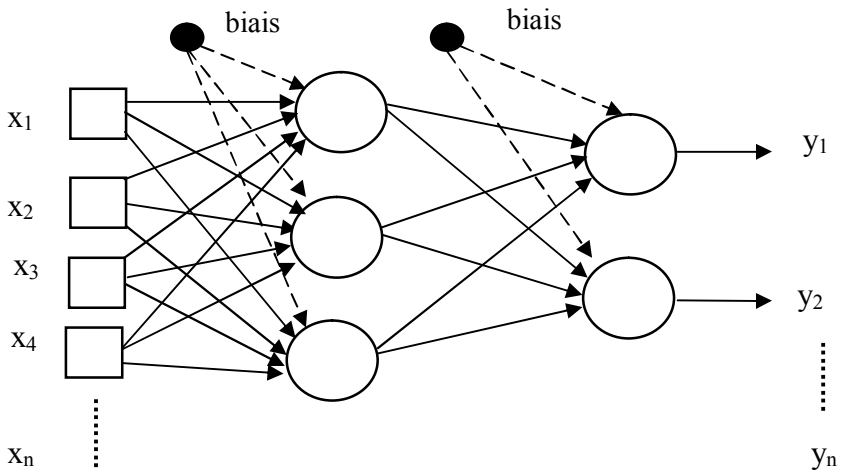


Figure III.8 : réseaux MLP à une couche cachée

Le graphe d'un réseau de neurones non bouclé est acyclique (figure III.8) : si l'on se déplace dans le réseau, à partir d'un neurone quelconque, en suivant les connexions on ne peut pas revenir au neurone de départ.

Les entrées (représentées par des carrés sur la figure III.8) ne sont pas des neurones : elles ne réalisent aucun traitement de l'information, puisqu'elles ne font que transmettre les valeurs des variables. (On voit souvent des textes qui mentionnent les entrées comme couche d'entrée ou neurone d'entrée, alors que cette expression est trompeuse).

Formellement, soit X un vecteur d'entrée tel que :$X = (x_1, x_2, \ldots..x_n)$, la propagation avant permet de récupérer sur la couche de sortie un vecteur $Y = (y_1, y_2 \ldots\ldots..y_m)$.Donc, un MLP défini une fonction de $\mathcal{R}^n$ dans $\mathcal{R}^m$.

Pour un MLP d'architecture *fixé*, la fonction définie par le réseau dépend des valeurs des poids **W** de ses différentes connexions. Une architecture génère donc une famille de fonctions :

$$\mathcal{R}^n \longrightarrow \mathcal{R}^m$$

$$X \longrightarrow Y = \mathbf{g}(X, W)$$

L'ensemble des familles définies par l'ensemble des architectures possibles est très adapté à la recherche de fonctions de régression non-linéaires.

III.6.1 Propriétés d'approximations du MLP

La propriété fondamentale des réseaux MPL est qu'ils sont des approximateurs universels. En effet, Il a été mathématiquement démontré qu'un réseau MLP composé d'une seule couche cachée avec pour fonction d'activation une fonction sigmoïde et d'une couche de sortie linéaire, peut approcher bien toute fonction multi-variable continue avec une précision arbitraire [CYB 89, FUN 89, HOR 89, XIA 05].

C'est que justifie l'utilisation *réelle* des réseaux MLP : la recherche d'une fonction de régression à partir d'un nombre *fini* de points.

Par ailleurs, un réseau à deux couches cachées peut mieux rapprocher toute fonction continue, mais ceci va présenter des minimums locaux supplémentaires [CHE 90, XIA 05].

III.7 Les réseaux RBF

Les réseaux de neurones à fonctions radiales de base *(Radial Basis Function Network)* sont également des réseaux à propagation avant (*feedforwrd*). Ils possèdent une seule couche cachée composée de fonctions noyaux (φ) généralement gaussiennes. Les paramètres de cette dernière sont attachés à la non-linéarité du neurone, et non pas aux entrées du neurone.

Deux paramètres caractérisent ces fonctions noyaux [DU 06] :

- Un vecteur de référence μ_j appelé centre ou prototype ;
- La dimension σ_i du champ d'influence appelé rayon d'influence.

Chaque neurone de la couche cachée est caractérisé par la distance du vecteur d'entrée x au vecteur prototype μ_i : $r_i(x) = ||x - \mu_i||$.

Donc, La fonction gaussienne : $\varphi_i(x) = \exp\left(\frac{-r_i(x)^2}{\sigma_i^2}\right)$

Chaque neurone de sortie (linéaire) réalise une somme pondérée des sorties des neurones cachés.

La différence pratique entre les MLP et les RBF est la suivante : les RBF ont des non-linéarités *locales,* qui tendent vers zéro dans toutes les directions de l'espace des entrées ; leur zone d'influence est donc limitée dans l'espace, ce qui n'est pas le cas des neurones à fonction d'activation sigmoïde (MLP).

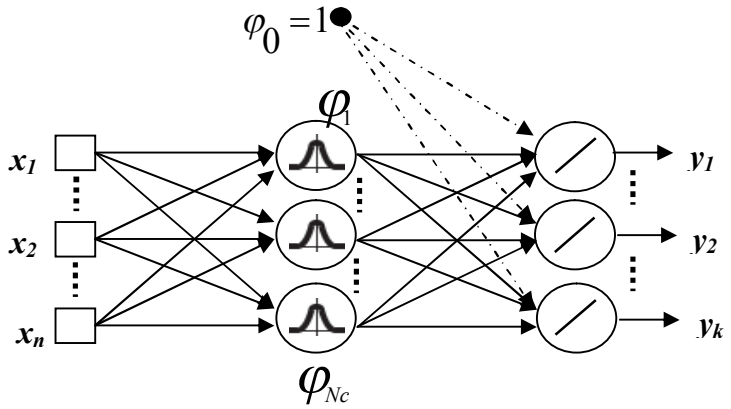


Figure III.9 : Réseau RBF.
$\varphi_0(x) = 1$ Correspond au biais de la couche de sortie

III.8 L'apprentissage des réseaux de neurones

On appel « apprentissage » des réseaux de neurones la procédure qui consiste à estimer les paramètres des neurones du réseau, afin que celui-ci remplisse au mieux la tâche qui lui est affectée [DRE 02].

On entend par paramètres les poids synaptiques (w_{ij}), qui sont ajustés plusieurs fois selon un processus graduel, itératif avant d'atteindre leurs valeurs finales. Cet ajustement a pour but de minimiser l'erreur en sortie. Ce processus itératif s'arrêtera soit en trouvant une solution optimale, soit en atteignant un nombre d'itérations fixé à l' avance.

On peut distinguer deux types d'apprentissages : l'apprentissage supervisé et l'apprentissage non supervisé.

III.8.1 L'apprentissage supervisé

Un réseau de neurone non bouclé réalise une relation non-linéaire Y=**g**(X, W) comme modèle de régression. On cherche à estimer les paramètres d'un réseau de neurones qui réalise une approximation de la fonction de régression, celle-ci est inconnue.
On connaît, en tous points ou seulement en certains points, les valeurs que doit avoir la sortie (Y) du réseau en fonction des entrées(X) correspondantes : c'est en ce sens que l'apprentissage est *supervisé*.

Donc, on cherche la fonction de régression (E(y/x) : espérance mathématique des valeurs observées *y* au point *x*) et, comme le nombre de point est fini, on ne peut y trouver qu'une approximation.

Pour trouver cette approximation, il faut définir une fonction de coût qui mesure l'écart entre la sortie du modèle (fonction réalisée par le réseau de neurone) et la sortie désirée. Plus la valeur de la fonction de coût est petite, plus le modèle reproduit fidèlement les observations utilisées pour l'apprentissage. L'algorithme d'apprentissage cherche donc à trouver le point, dans l'espace des paramètres, pour lequel la fonction de coût est minimale.

L'apprentissage des réseaux de neurones non bouclé se ramène donc au problème de la minimisation d'une fonction de coût (i.e. *Un problème d'optimisation sans contraintes).*

La procédure d'apprentissage supervisé nécessite :

1. *Un ensemble d'exemple d'apprentissage* ;
2. *La définition d'une fonction de coût qui mesure l'écart entre les sorties du réseau de neurones et les sorties désirées ;*
3. *Un algorithme de minimisation de la fonction de coût par rapport aux paramètres.*

III.8.1.1 L'ensemble d'exemple d'apprentissage

C'est un ensemble de couples d'observations {(entrée, sortie)}, ces couples d'observations (x_i , y_i), $i=1....N$, sont souvent des représentations bruitées ou ambigües d'une réalité sous-jacente.
Le nombre d'exemples de l'ensemble d'apprentissage a une grande importance.

En effet, comme la sortie du réseau est non linéaire par rapport aux paramètres, la fonction de coût peut présenter des minima locaux, et les algorithmes d'apprentissage ne donnent aucune garantie de trouver le minimum global [DU 06]. Toutefois**,** si le nombre

d'exemples est insuffisant, non seulement des minima locaux apparaissent, mais, le minimum global de la fonction de coût ne correspond pas forcément aux valeurs des paramètres recherchées. Par contre si l'on dispose d'un nombre suffisant d'exemples, le problème des minima locaux ne se pose pratiquement pas : il suffit, d'effectuer quelques apprentissages avec des initialisations différentes des paramètres.

III.8.1.2 Normalisation des entrées et des sorties

Avant tout apprentissage, il est indispensable de normaliser et de centrer toutes les variables d'entrées. En effet, si des entrées ont des grandeurs très différentes, celles qui sont « petites » n'ont pas d'influence sur l'apprentissage. Il est donc recommandé, pour chaque vecteur d'entrée $X_i = (x_1, x_2, \ldots..x_n)$, de calculer la moyenne $\boldsymbol{\mu_i}$ et l'écart type $\boldsymbol{\sigma_i}$ de ses composantes, et d'effectuer le changement de variable $\mathbf{x'_i = (x_i - \mu_i) / \sigma_i}$ [DRE 02].
La normalisation des sorties $Y_i = (y_1, y_2 \ldots\ldots..y_m)$ s'effectue avec la même technique.

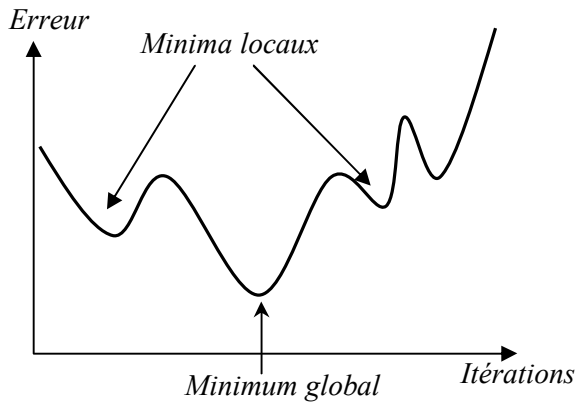


Figure III.10 : Exemple de minima locaux

III.8.1.3 Fonction de coût

La fonction de coût doit permettre de mesurer l'écart entre le modèle et les observations. Il existe un grand nombre de fonction [DU 06] ; nous présenterons la fonction du coût des *moindres carrés,* qui est la plus utilisée.

Pour un exemple *i* d'un ensemble d'observation E, la fonction de coût des moindres carrées est égale à la somme, sur les N_s neurones de la couche de sortie, des carrés des écarts entre la sortie du modèle (sortie du réseau de neurone= S^i) et la sortie désirée d^i. Comme la sortie du réseau dépend du vecteur de paramètre W, la fonction de coût en dépend également. On la note $J^i(w)$:

$$J^i(w) = \sum_{q=1}^{N_s} \left(d_q^i - S_q^i \right)^2 \quad (3.1)$$

Sur un ensemble d'exemple M la fonction de coût est notée $J^M(w)$ et elle est définie par la moyenne des carrées des écarts sur les N exemples de cet ensemble (Ecart Quadratique Moyen).

$$J^M(w) = \frac{1}{N} \sum_{i=1}^{N} J^i(w) \quad (3.2)$$

III.8.1.4 Algorithme d'optimisation

A partir d'une fonction de coût dépendant du vecteur de paramètres (w) et des exemples de l'ensemble d'apprentissage, il faut choisir l'algorithme d'optimisation qui permettra d'estimer le vecteur des paramètres pour le quel la fonction de coût choisie est minimale. De nombreux algorithmes ont été proposés : la rétropropagation de l'erreur, gradients conjugués, la méthode de levenberg-Marquardt, les méthodes par région de confiance, les filtres de kalman… [DU 06] **.**Nous n'aborderons ici que les méthodes les plus classiques.

Le principe de ces méthodes est de se placer en un point, de trouver une direction de descente du coût dans l'espace des paramètres (w), et ensuite, de se déplacer d'un *pas* suivant cette direction. On atteint un nouveau point et l'on recommence la procédure. On poursuit cette démarche jusqu'à satisfaction d'un critère d'arrêt.

Les méthodes d'optimisation non linéaires que nous présentons se différencient par le choix de la *direction de descente* et du *pas*.

Deux types de méthodes peuvent être mis en œuvre pour minimiser la fonction de coût :

- ✓ Des méthodes non adaptatives d'apprentissage (*batch training*) : elles consistent à estimer les paramètres du modèle par minimisation de la fonction de coût des moindres carrés, qui tient compte simultanément de tous les exemples de l'ensemble d'apprentissage ; l'utilisation d'une telle méthode nécessite que les N exemples soient disponibles dès le début de l'apprentissage ;
- ✓ Des méthodes adaptatives d'apprentissage (*online training*) : elles consistent à modifier les paramètres du modèle, successivement en fonction du coût partiel relatif à

chaque exemple k : $j^k(w) = (d_q^k - S_q^k)^2$, car les exemples sont présentés au réseau de neurone un par un. Cette méthode est appelée souvent méthode incrémentale. Cette technique est utilisée lorsqu'on désir effectuer l'apprentissage au fur et à mesure de la disponibilité des exemples.

L'objectif est d'avoir une erreur quadratique minimum sur les points de l'ensemble d'apprentissage, et non pas une erreur nulle. Du fait que, les mesures étant entachées de bruit. Un modèle parfait reproduirait le bruit et les perturbations, alors que l'objectif de la modélisation est, bien au contraire, d'extraire tout ce qui est déterministe et prédictible dans l'information présente dans les données. Nous aborderons plus en détails dans la section III.9 les problèmes liés à la généralisation.

III.8.1.5 Calcul du gradient par rétropropagation de l'erreur

C'est l'algorithme le plus connu, établi par [RUM 86], qui permet d'utiliser les méthodes d'optimisation non-linéaire basées sur le gradient. Il est non seulement employé pour les réseaux non bouclés FNNs tel que le MLP mais, il a été également adapté aux réseaux bouclés RNNs [PIN 87].

Soit (x, y) un exemple de la base d'apprentissage. Le neurone i calcul une grandeur y_i qui est une fonction non linéaire de son potentiel v_i ; le potentiel v_i est une somme pondérée des entrées x_j, la valeur de l'entrée x_j étant pondérée par un paramètre w_{ij} :

$$y_i = f\left(\sum_{j=1}^{n_i} w_{ij} x_j\right) = f(v_i)$$

Les n_i entrées du neurone i peuvent être soit les sorties d'autres neurones, soit les entrées du réseau. Dans toute la suite, x_j désignera donc indifféremment soit la sortie y_j du neurone j, soit l'entrée j du réseau.

L'objectif est de calculer le gradient de l'équation (3.2). Or pour évaluer le gradient, il suffit d'évaluer le gradient du coût partiel $j^k(w)$ relatif à l'observation k, et de faire ensuite la somme sur tous les exemples.

L'algorithme de rétropropagation consiste essentiellement en l'application répétée de la règle de dérivées composées. On remarque tout d'abord que la fonction de coût partielle ne dépend du paramètre w_{ij} que par l'intermédiaire de la valeur de la sortie du neurone i, qui est elle-même fonction uniquement du potentiel du neurone i ; donc :

$$\left(\frac{\partial j^k}{\partial w_{ij}}\right)_k = \left(\frac{\partial j^k}{\partial v_i}\right)_k \left(\frac{\partial v_i}{\partial w_{ij}}\right)_k = \delta_i^k x_j^k \qquad (3.3)$$

où :

$\left(\frac{\partial j^k}{\partial v_i}\right)_k$ Désigne la valeur du gradient du coût partiel par rapport au potentiel du neurone *i* lorsque les entrées du réseau sont celles qui correspondent à l'exemple *k.*

$\left(\frac{\partial v_i}{\partial w_{ij}}\right)_k$ Désigne la valeur de la dérivée partielle du potentiel du neurone *i* par rapport au paramètre w_{ij} lorsque les entrées du réseau sont celles qui correspondent à l'exemple *k*.

x_j^k est la valeur de l'entrée *j* du neurone *i* lorsque les entrées du réseau sont celles qui correspondent à l'exemple *k.*

Les quantités δ_i^k sont calculées d'une manière récursive, en menant les calculs depuis les sorties du réseau vers ses entrées.

En effet :

- Pour le neurone de sortie i :

$$\delta_i^k = \left(\frac{\partial j^k}{\partial v_i}\right)_k = \left(\frac{\partial [(y_p^k - g(x,w))^2]}{\partial v_i}\right)_k = -2\left[y_p^k - g\left(x^k, w\right)\right]\left(\frac{\partial g(x,w)}{\partial v_i}\right)_k$$

Or, la sortie du modèle est la sortie y_i du neurone de sortie ; cette relation s'écrit donc :

$$\delta_i^k = -2\left[y_p^k - g\left(x^k, w\right)\right] f'\left(v_i^k\right) \qquad (3.4)$$

où $f'\left(v_i^k\right)$ désigne la dérivée de la fonction d'activation du neurone de sortie lorsque les entrées du réseau sont celles de l'exemple k.

- Pour un neurone caché i :

La fonction du coût ne dépond du potentiel du neurone *i* que par l'intermédiaire des potentiels des neurones *m* qui reçoivent la valeur de la sortie du neurone *i,* c'est-à-dire de tous les neurones qui, dans le graphe de connexions du réseau, sont adjacents au neurone i, entre ce neurone et la sortie :

$$\delta_i^k \equiv \left(\frac{\partial j^k}{\partial v_i}\right)_k = \sum_m \left(\frac{\partial j^k}{\partial v_m}\right)_k \left(\frac{\partial v_m}{\partial v_i}\right)_k = \sum_m \left(\frac{\partial j^k}{\partial v_m}\right)_k \left(\frac{\partial v_m}{\partial v_i}\right)_k$$

Or $v_m^k = \sum_i w_{mi} x_i^k = \sum_i w_{mi} f(v_i^k)$ d'où $\left(\frac{\partial v_m}{\partial v_i}\right)_k = w_{mi}\ f'(v_i^k)$.

Finalement, on obtient la relation :

$$\delta_i^k = \sum_m \delta_m^k w_{mi} f'(v_i^k) = f'(v_i^k) \sum_m \delta_m^k w_{mi} \qquad (3.5)$$

Donc, l'algorithme de rétropropagation comporte deux phases pour chaque exemple *k* :

- Une phase de propagation, au cours de laquelle les entrées correspondant à l'exemple *k* sont utilisées pour calculer les sorties et les potentiels de tous les neurones.

- Une phase de rétropropagation, au cours de laquelle sont calculées les quantités δ_i^k. une fois que ces quantités sont disponibles, on calcul les gradients des coûts partiels par la relation (3.3), puis il suffit de faire la somme pour obtenir le gradient de la fonction « coût total ».

III.8.1.6 Modification des paramètres

Après évalué le gradient de la fonction de coût par rapport aux paramètres du modèle, on effectue une modification des poids, afin d'approcher d'un minimum de la fonction de coût. Nous allons citer quelques algorithmes, les plus couramment utilisées, de minimisation itérative des paramètres du modèle.

- **La descente du gradient**

La méthode du gradient consiste à modifier les paramètres par la formule suivante, à l'itération *i* de l'apprentissage :

$$w_i = w_{(i-1)} - \eta_i \nabla j(w_{(i-1)}) \quad \text{avec} \quad \eta_i > 0 \qquad (3.6)$$

C'est une méthode de descente itérative qui à chaque itération déplace la solution dans la direction la plus forte pente, c'est-à-dire la direction opposé au gradient. La descente du gradient converge vers un minimum local de l'erreur. La quantité η_i est appelé *pas du gradient* ou *pas* d'apprentissage.

Plusieurs heuristiques ont été proposées pour déterminer la valeur de η_i :

- Un pas d'apprentissage constant, $\eta_i = \eta_0$,
- Un pas adaptatif qui est augmenté lorsque la solution courante diminue l'erreur et réduit dans le cas contraire.

Pour accélérer la convergence de la méthode du gradient, [PLA 86] proposent d'ajouter un terme d'inertie dans la formule de mise à jour des poids :

$$\Delta w_t = -\eta_t \nabla j(w_t) + \mu \, \Delta w_{t-1} \tag{3.7}$$

Le scalaire μ est appelé moment (*momentum*), qui prend ses valeurs dans [0,1] (typiquement μ=0,1).

- **Les gradients conjugués**

La méthode des gradients conjugués [HES 52, Mol 93] consiste à choisir une direction conjuguée à la direction de la descente du gradient :

$d_t = -\nabla j(w_{(t)}) + \beta_t d_{t-1}$ avec d_{t-1} direction de descente du gradient.

$$w_{t+1} = w_t + \eta_t d_t \tag{3.8}$$

Où le pas d'apprentissage η_t est le résultat de la recherche linéaire dans la direction d_t et où β_t est déterminé généralement par la formule de Polak-Ribière [MAH 05]:

$$\beta_t = \frac{g_t^T (g_t - g_{t-1})}{g_{t-1}^T g_{t-1}} \qquad \text{où } g_t = \nabla j(w_{(t)})$$

Cette formule permet de réinitialiser d_t à la direction de plus forte pente lorsque l'algorithme ne progresse plus (lorsque $g_t - g_{t-1}$ devient négligeable).

On peut remarquer que la descente du gradient avec *momentum* est une approximation de la méthode des gradients conjugués. Or, avec la descente du gradient l'utilisateur doit choisir empiriquement les valeurs du pas d'apprentissage et du *momentum*, celles-ci sont déterminées automatiquement pendant l'apprentissage en utilisant la méthode des gradients conjugués.

- **Levenberg-Marquardt**

La méthode de Levenberg-Marquardt [LEV 44, MAR 63] consiste à modifier les paramètres selon la relation suivante :

$w_k = w_{k-1} - [H_{k-1}(j(w_{k-1})) + \lambda_{k-1} . I]^{-1} \nabla j(w_{(k-1)})$

où I est la matrice identité qui assure l'ajout du terme de relaxation variable λ_{k-1} aux éléments de la diagonale de $H_{k-1}(j(wk-1))$. A chaque itération k le facteur de relaxation λ_{k-1} est incrémenté ou décrémenté. Ainsi, λ_{k-1} diminue à l'approche de l'optimum et augmente dans le cas contraire. Typiquement $\lambda_0 = 0,1$.

III.8.1.7 Résumé de la rétropropagation du gradient

Etape1 : initialisation des poids par des petites valeurs aléatoires.

Etape 2 : présentation d'un vecteur d'entrée et celui de la sortie désirée.

Etape 3 : calcul de façon itérative des états des neurones dans les couches suivantes

(k=1 à k) :

$$V_j^k = F(\sum_{i=1}^{n}(W_{ji}^k V_i^{k-1} + b_j^k))$$

Avec :

F **:** la fonction d'activation

k **:** Nombre de couches du réseau.

n **:** Nombre de neurones dans la couche k-1.

V_j^k **:** La sortie du neurone j de la couche k.

V_j^0 **:** Entrée du neurone j de la couche d'entrée.

W_{ji}^k **:** Connexion entre le neurone i de la couche k-1 et le neurone j de la couche k.

b_j : biais de neurone j de la couche k

Etape 4 : calcul du gradient de l'erreur faite sur le neurone i dans la couche de sortie :

$$\delta_i^k = 2.f'(\sum_{i=1}^{n}(W_{ji}^k V_i^{k-1} + b_i^k)).(D_i^k - V_i^k)$$

Où :

f' : La dérivée de la fonction F.

D_i^k : La sortie désirée pour le neurone i dans la couche k.

Etape 5 : calcul de gradient des erreurs rétropagées de la couche k à la couche 1.

$$\delta_i^{k-1} = f'(\sum_{i=1}^{n}(W_{ji}^k V_i^{k-1} + b_i^{k-1})).\sum_{i=1}^{n}\delta_i^k W_{ji}^k$$

Etape 6 : ajustement des poids

$$W_{ji}^k(t+1) = W_{ji}^k(t) - \eta\delta_j^{k-1}$$

Etape 7 : répétition de toutes les étapes, sauf la première, chaque fois pour un nouveau exemple, jusqu'à l'activation de test d'arrêt (convergence).

III.8.2 L'apprentissage non supervisé

C'est un apprentissage sans professeur, qui consiste à ajuster les poids à partir d'un seul ensemble d'apprentissage formé uniquement de données (les entrées). Aucun résultat désiré n'est fourni au réseau. Le réseau va de lui même catégoriser les variables d'entrées ; une sorte d'analyse de données : on cherche à regrouper les données, selon des critères de ressemblance qui sont inconnus a priori.

Les réseaux de neurones à apprentissage non supervisé les plus utilisés sont les cartes auto-organisatrices (SOM, Self-Organization Map) [KOH 82].

III.9 Apprentissage et généralisation

La méthode de minimisation de la fonction de coût conduit à concevoir des modèles de complexités différentes et à choisir celui qui est susceptible d'avoir les meilleures propriétés de généralisation ; c'est-à-dire fournir un modèle capable de prédire la valeur de la sortie sur de nouveaux exemples.

Le meilleur modèle réalise un compromis entre la précision de l'apprentissage et la qualité de la généralisation. Si le modèle sélectionné est trop peu complexe, l'apprentissage et la généralisation sont peu précis ; si le modèle est trop complexe, l'apprentissage est satisfaisant, mais la généralisation ne l'est pas, ainsi le modèle est surajusté. Ce compromis entre la qualité de l'apprentissage et celle de la généralisation, gouverné par la complexité du modèle, est connu sous le terme de dilemme *biais-variance* [GEM 92] : Quand la complexité

du modèle augmente, le biais diminue (le modèle "colle" mieux aux données) mais la variance augmente (le modèle change beaucoup lorsqu'on considère une autre base d'apprentissage). Le terme *biais* désigne la moyenne sur toutes les bases d'apprentissages possibles du carré de la différence entre les prédictions du modèle et de la fonction de régression. Tandis que la *variance* reflète l'influence du choix de la base d'apprentissage sur le modèle.

III.9.1 Généralisation par un arrêt prématuré

L'arrêt prématuré (*early stopping*) consiste à arrêter l'apprentissage avant la convergence complète de l'algorithme. Ainsi, le modèle ne s'ajuste pas trop finement aux données d'apprentissage : le surajustement est limité. La méthode la plus classique pour choisir le moment d'arrêter l'apprentissage, consiste à suivre l'évolution de la fonction de coût sur une base de validation (une base non utilisée pour l'apprentissage), et à arrêter les itérations lorsque le coût calculé sur cette base commence à croître. (Figure III.11)

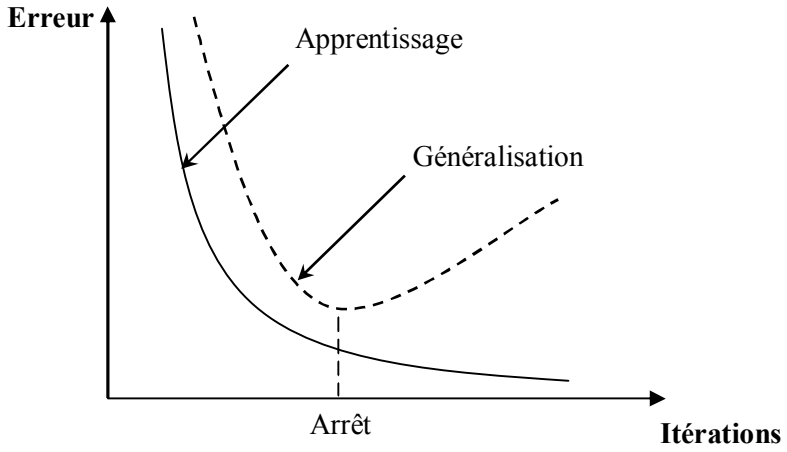


Figure III.11 : l'évolution de la fonction de coût [DU 06].

III.9.2 Généralisation par régularisation

Les méthodes de régularisation sont des méthodes de pénalisation, qui consistent à ajouter un terme à la fonction de coût usuelle afin de favoriser les fonctions régulières. La fonction à minimiser est alors de la forme : $J' = J + \alpha.\Omega$ où j est, par exemple, la fonction de coût des moindres carrées. La fonction Ω la plus largement utilisée est celle qui pénalise les modèles ayant des poids élevés : $\Omega = \sum_i \|w_i\|^2$ [DRE 02].

L'apprentissage est réalisé en minimisant la nouvelle fonction *J'*. Un modèle qui a bien appris la base d'apprentissage correspond à une valeur faible de *J*. la méthode du *weight decay* est souvent utilisée, car elle est simple à mettre en œuvre, et plusieurs études ont montré qu'elle conduisait à de bons résultats (voir par exemple [REE 95], [DU 06])

III.9.3 Régularisation par modération des poids (Weight-decay)

Lorsque les poids du réseau sont grands en valeur absolue, les sigmoïdes des neurones cachés sont saturées, si bien que les fonctions modélisées peuvent avoir des variations brusques. Pour obtenir des fonctions régulières, il faut travailler avec la partie linéaire des sigmoïdes, ce qui implique avoir des poids dont la valeur absolue est faible.

La méthode de régularisation du *weight decay* limite la valeur absolue des poids en utilisant $\Omega = \frac{1}{2}\sum_{i=1}^{q} w_i^2$. Donc, $J' = J + \frac{\alpha}{2}\sum_{i=1}^{q} w_i^2$ où q est le nombre de paramètres du réseau, et α est un *hyperparamètre* dont la valeur doit être déterminée par un compromis : Si α est trop grand, les poids tendent rapidement vers zéro, le modèle ne tient plus compte des données. Si α est trop petit, le terme de régularisation perd de son importance et le réseau de neurones peut donc être surajusté.

Le principe de la mise en œuvre de la méthode est simple : on calcule le gradient de la fonction de coût J par rétropropagation, puis en lui ajoute la contribution du terme de régularisation : $\nabla J' = \nabla J + \alpha w$

Pour tenir compte du caractère différent des poids en fonction des couches, il faut considérer plusieurs hyperparamètres :

$$J' = J + \frac{\alpha_0}{2}\sum_{W_0} w_i^2 + \frac{\alpha_1}{2}\sum_{W_1} w_i^2 + \frac{\alpha_1}{2}\sum_{W_2} w_i^2$$

Où W_0 représente l'ensemble des paramètres reliant les biais aux neurones cachés, où W_1 représente l'ensemble des poids reliant les paramètres aux neurones cachés, et W_2 l'ensemble des paramètres reliés au neurone de sortie (y compris le biais du neurone de sortie). Le modèle comprend trois hyperparamètres α_1, α_2, α_3 qui doivent être déterminés. Une démarche heuristique, consiste à effectuer plusieurs apprentissages avec des valeurs différentes des paramètres, à tester les modèles obtenus sur un ensemble de données de validation, et à choisir le meilleur.

III.10 Sélection de modèle

Comme indiqué dans la section précédente, la sélection de modèles est une étape cruciale dans la conception d'un modèle par apprentissage. Nous allons présenter les méthodes les plus fréquemment utilisées :

III.10.1 La validation croisée (cross-validation)

La validation croisée est une méthode d'estimation de l'erreur de généralisation d'un modèle à partir de données qui ne sont pas utilisées en apprentissage [JAN 88]. On divise l'ensemble des données disponible en D sous ensembles disjoints. On effectue ensuite les étapes suivantes :

1. itération i à réaliser D fois : construire une base d'apprentissage par l'union de D-1 sous-ensemble ; effectuer plusieurs apprentissages, avec des initialisations différentes des poids, avec cette base d'apprentissage ; pour chacun des modèles obtenus, calculer l'erreur quadratique moyenne sur l'ensemble de validation (EQMV) par le sous ensemble des N_v exemples restants $E_V = \sqrt{\frac{1}{N}\sum_{k=1}^{N_V}(y_k^p - g(x_k, w))^2}$, retenir la plus petite de ces erreurs E_{vi}.

2. calculer le score de validation croisée à partir des D quantités E_{vi} au cours de ces D itérations $\sqrt{\frac{1}{D}\sum_{i=1}^{D}(E_{vi})^2}$; ce score constitue une estimation de l'erreur de généralisation de la famille de modèle considéré.

Ainsi, si l'on choisit D=5 (ce qui est une valeur typique appelée en anglais *5-fold cross-validation*) ; on réalise 5 partitions différente de la base de données, dans chacune de ces partitions 80% des données sont dans l'ensemble d'apprentissage et 20 % dans l'ensemble de validation (figure III.10). On sélectionne le modèle ayant le plus faible score de validation.

Cette technique a l'avantage d'être moins sensible au choix des bases de validation, et permet d'utiliser toutes les données en apprentissage. Elle nécessite cependant d'effectuer D apprentissages

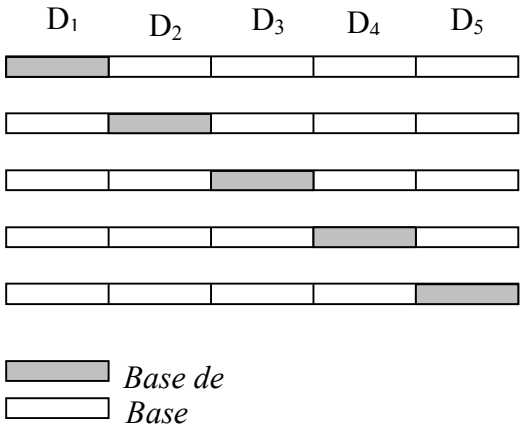


Figure III.12 : Principe de la validation croisée

III.10.2 La méthode Leave-one-out

L'estimation de l'erreur de généralisation par leave-one-out est un cas particulier de la validation croisée, pour laquelle D=N : à l'itération *k* de l'ensemble d'apprentissage, on effectue des apprentissages (avec des initialisations différentes des poids) avec les N-1 élément de la base d'apprentissage ; pour chacun des modèles obtenus, on calcule l'erreur de prédiction commise sur l'observation *k* lorsque celle-ci est extraite de l'ensemble d'apprentissage, et l'on retient la plus petite de ces erreurs, notée $R_k^{(-k)}$. Le score de leave-one-out est alors défini par $E_t = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(R_k^{-k})^2}$. On utilise ce score, comme dans le cas de la validation croisée, en augmentant progressivement la complexité des modèles.

III.11 Conclusion

Nous avons présenté dans ce chapitre les réseaux de neurones MLP comme outils pour la régression non-linéaire. Nous avons discuté de l'apprentissage de ces modèles paramétriques et du problème de la généralisation lié au principe de minimisation du risque empirique. Nous avons présenté notamment les méthodes classiques pour augmenter la généralisation d'un modèle d'apprentissage.

Ces réseaux sont maintenant considérés comme des outils mathématiques, indépendamment de toute référence à la biologie ; permettant de résoudre, avec une grande précision, les problèmes de modélisation ou d'identification linéaires et non-linéaires.

Chapitre IV

Mise en œuvre

IV.1 Introduction

Nous allons construire un modèle neuronal basé sur le modèle de file d'attente M/G/1/K PS (voir section II.3.2). Le modèle recherché ne vise pas à reproduire le fonctionnement interne du serveur web mais uniquement ses performances. Autrement dit, le serveur web peut être vu comme une boîte noire pour laquelle on dispose des mesures. En entrée les directives (paramètres) du serveur et en sortie les critères quantitatifs de la Qds (figure IV.1).

L'objectif est de reproduire les performances du serveur web à travers des mesures afin d'améliorer la Qds et de prédire les valeurs de ses critères.

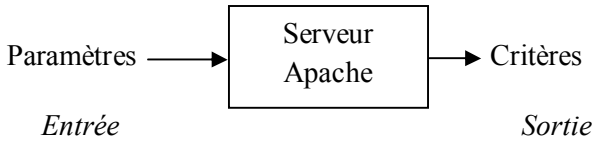


Figure IV.1 : Serveur web vu comme une boite noir

IV.2 Les entrées du modèle (paramètres)

IV.2.1 Le Trafic d'arrivé

Une session de navigation est un enchainement de requêtes pour les documents web : un usager commence une session sur un serveur en demandant un premier document, après l'avoir lu il décide de suivre un autre hyperlien qui le mène vers un autre document et ainsi de suite jusqu'à ce que l'usager décide de ne plus suivre aucun hyperlien dans le même serveur.

Le trafic total est le résultat de l'addition des requêtes produites par chaque session. Selon une étude faite par Google [GOO 10], ce sont des milliards de pages web qui sont consultés par les utilisateurs chaque mois (voir Annexe C). Nous allons donc présenter au réseau de neurones le nombre de connexions par seconde comme premier paramètre.

[YEU 99, AND 05b, AND 05a] ont monté que le trafic d'arrivée des requêtes suit une loi de poisson. La figue 4.1 montre la distribution du nombre de requêtes par seconde comparé à la distribution de poisson, et ce, après une analyse des fichiers logs de plusieurs serveurs web.

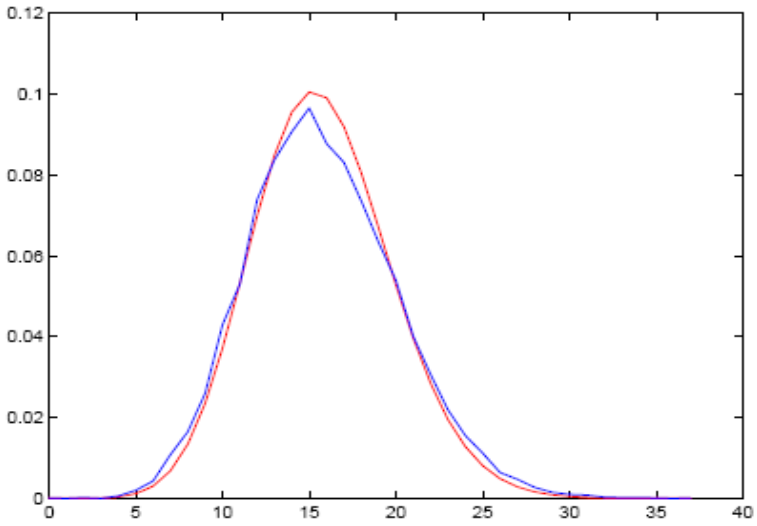


Figure IV.2 : La distribution du nombre de requêtes par seconde comparé à la distribution théorique de poisson [AND 05b].

IV.2.1.1 Classement des sites web les plus visités dans le monde

Une analyse statistique des dix premiers sites web les plus visités au monde permet d'avoir une image sur le trafic réel du web, cela nous donne une estimation du nombre de requêtes qu'il faut générer sur le serveur Apache et même amener le serveur Apache à être surchargé.

Le tableau 4.1 (voir Annexe D pour plus d'informations) récapitule l'analyse faite en juillet 2010 par des entreprises de statistique sur le web [ALE 10, COM 10]. Le nombre de visiteurs est estimé par [STA 10] et le Temps de réponse est calculé par des clients d'Alexa [ALE 10]**.**

IV.2.2 MaxClients

Le deuxième paramètre qui influe sur la Qds est *MaxClients* : La limite maximale des requêtes simultanées que le serveur peut prendre en charge (voir section II.2.4.1 pour le choix de ce paramètre).

IV.2.3 La taille de la file d'attente

Le troisième paramètre est la taille de la file d'attente du serveur web Apache, qui correspond à la directive **Listenbacklog** (voir section II.2.4.2). Néanmoins, Linux, réduit la taille de la *backlog* à la valeur de *somaxconn* si elle lui est supérieure (voir section II.2.5).

Pour qu'il n'y ait pas de conflit entre la valeur de *Listenbacklog* et la valeur de *somaxconn,* nous avons décidé que, pour chaque mesure de critère de performance, *Listenbacklog* doit être égale à *somaxconn* . Cela, nous a permis de prendre en compte un des paramètres du noyau du système d'exploitation sur lequel est exécuté le serveur web.

classement	**site web**	**Nombre de Visiteurs unique par jour**	**Temps moyen de réponse (seconde)**	**Taille de la page d'accueil (octets)**
1	www.google.com	118 405 724	1.4	13017
2	www.yahoo.com	100 199 679	1.735	134810
3	www.youtube.com	73 316 834	1.499	112588
4	www.facebook.com	73 323 054	1.711	95728
5	www.live.com	56 850 041	2.393	14667
6	www.msn.com	25 038 896	2,45	151846
7	www.wikipedia.org	23 763 575	0.987	84618
8	www.blogger.com	18 913 362	1.718	75714
9	www.baidu.com	13 642 305	2.29	13702
10	www.qq.com	12 079 254	2.193	186376

Tableau 4.1 : Analyse statistique des dix premiers sites web les plus visités au monde [ALE 10] [COM 10].

IV.3 Les sorties du modèle (critères)

Notre modèle neuronal est basé sur le modèle de file d'attente M/G/1/K PS (le choix est justifié dans la section I.12.2), ainsi on reproduit les mêmes critères quantitatifs de Qds : le débit, le temps moyen de réponse et le taux de rejet.

a) Le débit : le nombre maximum des requêtes que le serveur peut traiter par seconde**.**

b) Le temps de réponse : le temps moyen requis pour que le serveur exécute une requête. La latence causée par la liaison entre les clients et le serveur est négligeable. Si le temps de réponse est élevé le serveur est considéré surchargé.

c) Le taux de rejet : On parle de taux de rejet lorsque le taux d'arrivée excède la capacité du serveur ; Pratiquement, c'est le rapport entre le nombre de requêtes non admises par le serveur

et le nombre total de requêtes soumises au serveur pendant une période de temps. Si le taux de rejet =0 le serveur est disponible si non il est surchargé.

IV.4 Construction de la base d'apprentissage

On désire établir un modèle neuronal d'une dépendance entre les métriques de performances et les paramètres du serveur web. Pour cela, il faut construire une base d'exemple {(entrée, sortie)} figure IV.2.

Dans un premier temps, il faut avoir un outil qui génère les requêtes suivant un processus de poisson, vu que le processus d'arrivée de requêtes est poissonnien (voir section IV.2.1). Ensuite, il faut mesurer les critères de performance.

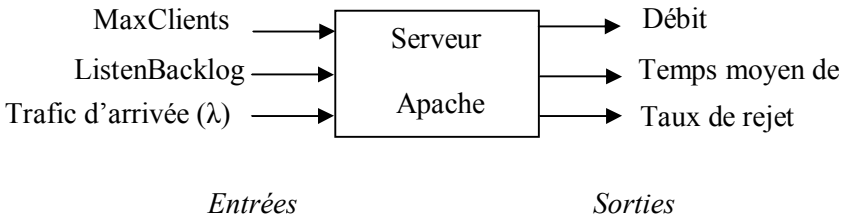


Figure IV.3 : Technique de construction de la base d'apprentissage

IV.4.1 Les outils de mesure et d'évaluation des performances

Il existe plusieurs outils de mesure et d'évaluation de performances d'un serveur web : SURGE [BRA 98], SpecWeb96 [SPE 96], HTTPERF [MOS 98]. Ces outils sont utiles pour l'analyse et la prédiction des performances d'un serveur web. Dans le cadre de notre recherche nous avons choisi HTTPERF comme logiciel de benchmark.

IV.4.1.1 HTTPerf

HTTPerf [MOS 98] est un outil d'évaluation permettant de générer des requêtes suivant le processus de poisson. Il est possible de créer des sessions, chacune envoyant un certain nombre de requêtes par seconde. Ceci pouvant alors nous simuler au mieux la réalité concernant les flux HTTP sur internet en direction d'un serveur web.

A la fin de chacune des simulations, httperf génère un rapport indiquant les temps de réponse pour les requêtes, le nombre de requêtes envoyées par seconde, le nombre de requêtes satisfaites…etc.

Exemple

httperf --server hostname \--port 80 --uri /test.html \--rate 150 --num-conn 27000 \

--num-call 1

Dans cet exemple httperf génère un flux de 27000 connexions TCP avec un taux de 150 connexions par seconde vers le serveur web *hostname*. *num_call* indique le nombre de requêtes http par connexion.

Le rapport généré par httperf à la fin de simulation :

Total: connections 27000 requests 26701 replies 26701 test-duration 179.996 s

Connection rate: 150.0 conn/s (6.7 ms/conn, <=47 concurrent connections)

Connection time [ms]: min 1.1 avg 5.0 max 315.0 median 2.5 stddev 13.0

Connection time [ms]: connect 0.3

Request rate: 148.3 req/s (6.7 ms/req)

Request size [B]: 72.0

Reply rate [replies/s]: min 139.8 avg 148.3 max 150.3 stddev 2.7 (36 samples)

Reply time [ms]: response 4.6 transfer 0.0

Reply size [B]: header 222.0 content 1024.0 footer 0.0 (total 1246.0)

Reply status: 1xx=0 2xx=26701 3xx=0 4xx=0 5xx=0

CPU time [s]: user 55.31 system 124.41 (user 30.7% system 69.1% total 99.8%)

Net I/O: 190.9 KB/s (1.6"10^6 bps)

Errors: total 299 client-timo 299 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Dans un premier temps, nous avons constaté que la cadence est insuffisante, nous ne retombions pas sur des résultats concluants pour la saturation du serveur.

Nous nous sommes alors penchés vers un autre outil : Autobench [AUT 10].

IV.4.1.2 Autobench

Il est souvent difficile voire même impossible de saturer un server web en l'évaluant d'une seule machine. En revanche, Autobench [AUT 10] permet à un nombre de machines d'être utilisées pour conduire l'évaluation synchronisée d'une cible en collectant automatiquement les résultats.

Autobench est constitué d'un serveur Autobenchd pour chaque client et d'un client Autobench_admin jouant le rôle d'un synchronisateur et collecteur des résultats. Chaque autobenchd lance un processus *Httperf* (voir paragraphe précédant) permettant de créer un stress au serveur web.

IV.4.2 Taille du document d'apprentissage

Pour simuler au mieux la réalité concernant les requêtes des utilisateurs du web, il faut tout d'abord, avoir un document (page web) proche de la réalité. Pour cela, nous avons analysé les objets des dix sites les plus populaires au monde (voir Annexe D). Nous avons calculé : La taille moyenne de la page d'accueil et le nombre moyen d'objets.

Par conséquent, nous avons créé une page dynamique php contenant 19 objets (le nombre moyen d'objets), avec une taille de 88307 octets (taille moyenne calculée).

IV.4.3 Configuration du laboratoire

a) Configuration optimale d'Apache

Grâce à notre étude sur le serveur web apache (voir chapitre 2), nous avons configuré Apache d'une manière optimale afin d'avoir les meilleurs résultats en terme de performance.

En effet, nous avons positionné **KeepAlive** à **on,** ce qui permet à un client d'effectuer plusieurs requêtes HTTP en séquence sur la même connexion.

Les directives relatives aux processus ont les valeurs :

```
StartServers  8
MinSpareServers  5
MaxSpareServers  20
MaxRequestsPerChild  4000
```

Le mode Prefork a été choisi car le "prefork" a l'avantage d'être très stable par rapport au mode worker.

La configuration du laboratoire est illustrée par la figure IV.3.

b) Le serveur

Pour que la base d'apprentissage soit liée à la réalité, nous avons choisi l'un des serveurs de l'université de Tiaret, qui est un PC Pentium dual core de 1,8 GHZ avec 1Go de RAM. Sa plateforme est Linux Debian muni d'un serveur web Apache 2.2.9 Prefork.

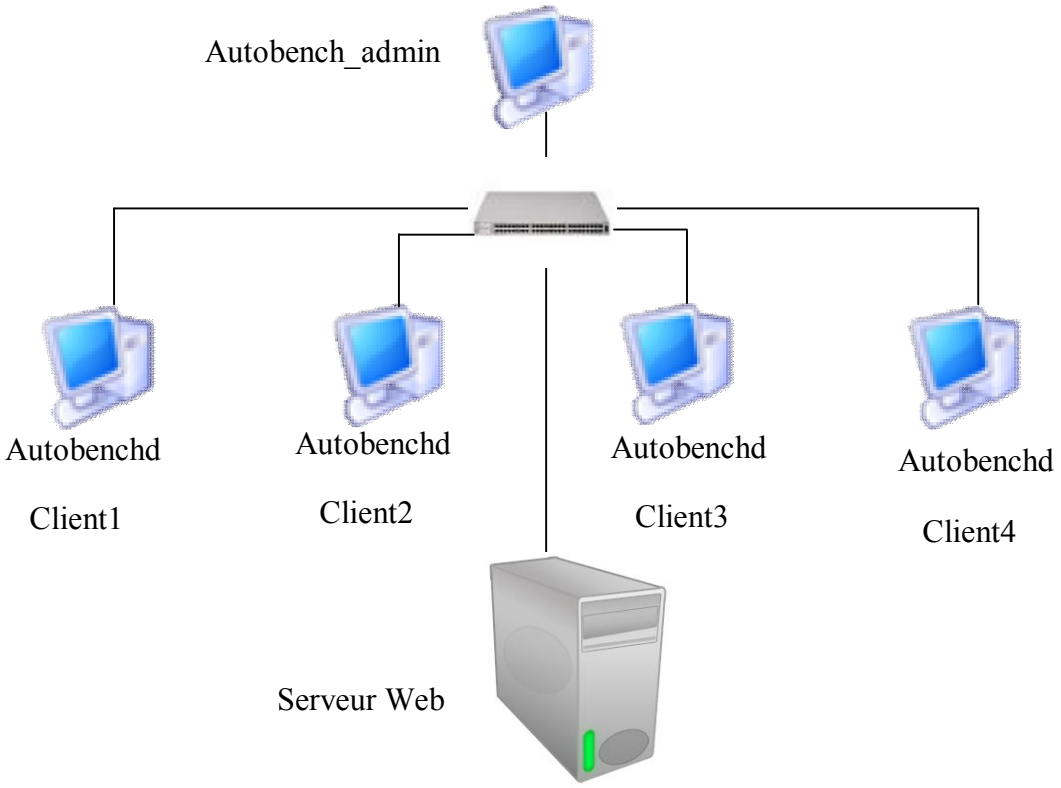


Figure IV.4 : La configuration du laboratoire

c) Les clients

Cinq PC : Pentium 4 de 1,8 GHZ , avec 512 Mo de RAM, plate forme :Linux ubuntu 9.4.

Nous avons installé *autobench_admin* sur une machine et *autobenchd* sur les 4 restantes.

Pour que la latence causée par la liaison entre les clients et le serveur soit négligeable les 5 machines ainsi que le serveur sont connectés à un commutateur CISCO CATALYST 2950 ,12 ports 10/100 Mbits.

Dans un premier temps, nous avons installé *autobenchd* sur deux machines, mais la surcharge du serveur web n'a pas été atteinte. Donc, nous avons augmenté le nombre de machines à quatre.

IV.4.4 Données d'apprentissage

Nous allons examiner les effets des paramètres du noyau de Linux et du serveur Apache sur les métriques des performances du serveur web. En effet, nous avons fait varier

les valeurs des trois paramètres : MaxClients de 25 à 300 avec un pas de 25 et la taille de la file d'attente (Listenbacklog =Somaxconn) de 25 à 500 avec un pas de 50.

Nous avons utilisé Autobench_admin pour lancer les tests de performances. Autobenchadmin permet de lancer l'outil de génération de trafic http httperf sur les 4 machines qui à leur tour émettent des requêtes suivant un processus de Poisson de paramètre λ. Nous avons fait varier λ de 25 à 500 avec un pas de 25. Pour chaque valeur de lambda, Autobench_admin récupère les métriques des performances : le temps moyen de réponse, le débit et le taux de rejets. Ceci nous a permis d'avoir une base d'apprentissage constituée de 4800 exemples.

Les valeurs et le pas de chaque paramètre ont été déterminés après plusieurs essais. Au début des tests nous avons choisi un pas égal à 100 avec un maximum de 1000, mais les métriques récupérées n'ont pas été en coordination avec nos attentes. L'objectif des tests est de définir à quel moment le serveur web entrera dans une phase de charge maximale, la phase où les requêtes des clients commencerons à prendre de plus en plus de temps à récupérer la réponse voire commencerons à être directement rejetées et par conséquent à ne plus satisfaire les critères de qualité de service perçus par les clients

- **Normalisation des entrées et des sorties**

Avant d'être présentées au MLP pour apprentissage, les données sont centrées et réduites $\mathbf{x'_i = (x_i - \mu_i) / \sigma_i}$ (voir section III.8.1.2).

IV.5 L'apprentissage

Comme nous l'avons mentionné, nous avons retenu le Perceptron Multicouches (MLP) comme base du modèle. Nous structurerons ce réseau en précisant le nombre de couches et de neurones cachés pour que le réseau soit en mesure de reproduire et surtout de généraliser ce qu'il a appris.

Nous avons choisi un réseau MLP à deux couches. La fonction d'activation des neurones cachés est la fonction tangente hyperbolique, et la fonction linéaire pour les neurones de sorties. (Ce choix est justifié dans la section III.6.1).

Le nombre de neurones de la couche cachée à été déterminé empiriquement en effectuant plusieurs essais. Nous nous somme efforcés de ne pas surdimensionner le MLP pour éviter l'apprentissage par cœur. Pour cela, nous avons augmenté progressivement la complexité du modèle de 10 jusqu'à 34 neurones (Figure IV.13). Nous avons constaté qu'au-

delà de 34 neurones, le réseau devient trop complexe et la capacité de généralisation devient faible.

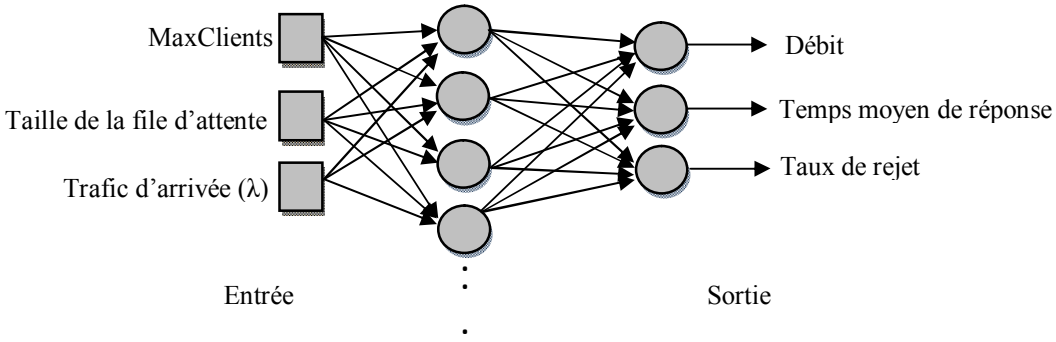


Figure IV.5 : Réseau MLP pour l'apprentissage des performances du serveur web

Nous avons calculé le gradient par rétropropagation de l'erreur. La méthode de descente du gradient a été utilisée avec un pas adaptatif pour modifier les poids. Ainsi, pour accélérer la convergence de la méthode nous avons ajouté un terme « momentum » (voir section III.8.1.6). Nous avons utilisé le toolbox de Matlab [MAT 10] pour réaliser l'apprentissage. Pour que le surajustement soit limité l'arrêt prématuré a été utilisé (voir section III.9.1).

Enfin, Pour évaluer l'erreur de généralisation, nous avons utilisé une méthode de validation croisée de type leave-k-out : la base d'exemples a été divisée en 5 parties de tailles égales, on a utilisé ensuite 4 parties pour l'apprentissage et une partie pour la validation du modèle. Ainsi, 80% des données ont été aléatoirement sélectionnées pour l'apprentissage et 20% pour valider le modèle. Ainsi, nous avons effectué 5 apprentissages pour chaque architecture avec des initialisations différentes. Parmi les modèles obtenus, nous avons retenu celui qui a fourni la plus faible erreur en validation.

Architecture	Nombre de paramètres	EQM $(\times 10^{-5})$
$3 \times 10 \times 3$	73	128,4
$3 \times 15 \times 3$	108	114,7
$3 \times 20 \times 3$	143	78,5
$3 \times 25 \times 3$	178	66,74
$3 \times 30 \times 3$	213	13,4
$3 \times 34 \times 3$	241	8,18

Tableau 4.1 : Erreurs de généralisation de différentes architectures de MLP

IV.6 Résultats et discussion

Le modèle obtenu après la phase d'apprentissage a été testé sur 20% des données qui n'ont pas été servi en apprentissage. Les figures d'IV.6 à IV.13 illustrent les différent cas des trois métriques de performance. Nous constatons que le modèle retenu généralise à 98% les données apprises.

Les figures (IV.6), (IV.7) et (IV.8) mettent en évidence la variation du temps moyen de réponse, le pourcentage du rejet et le débit du serveur en fonction du taux d'arrivée des requêtes http. Tandis que Les figures (IV.9), (IV.10) et (IV.11) montrent successivement les même courbes mais en fonction de la taille de la file d'attente (Listenbacklog). Alors que Les figures (IV.12), (IV.13) et (IV.14) montrent ces mêmes métriques en fonction de MaxClients.

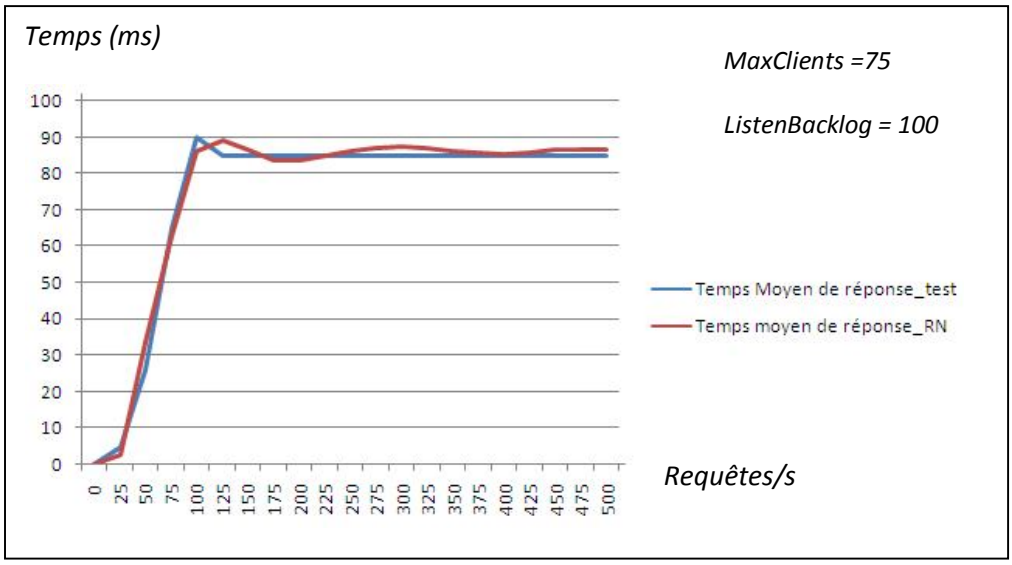


Figure IV.6 : Temps moyen de réponse par rapport du trafic d'arrivée

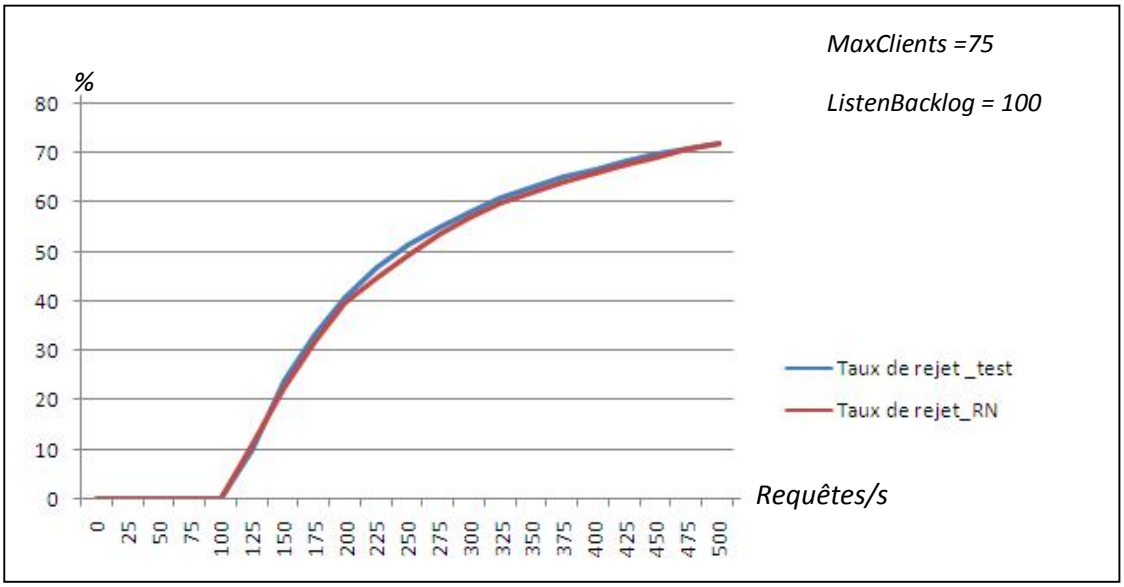


Figure IV.7 : Taux de rejet par rapport au trafic d'arrivée

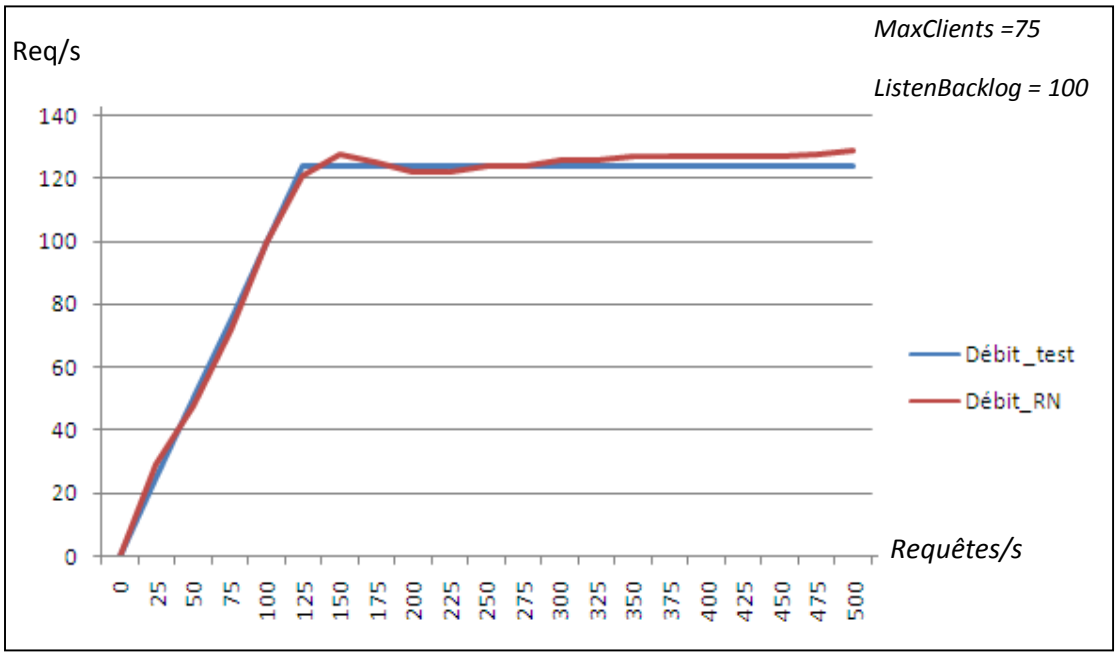


Figure IV.8 : Débit du serveur par rapport au trafic d'arrivée

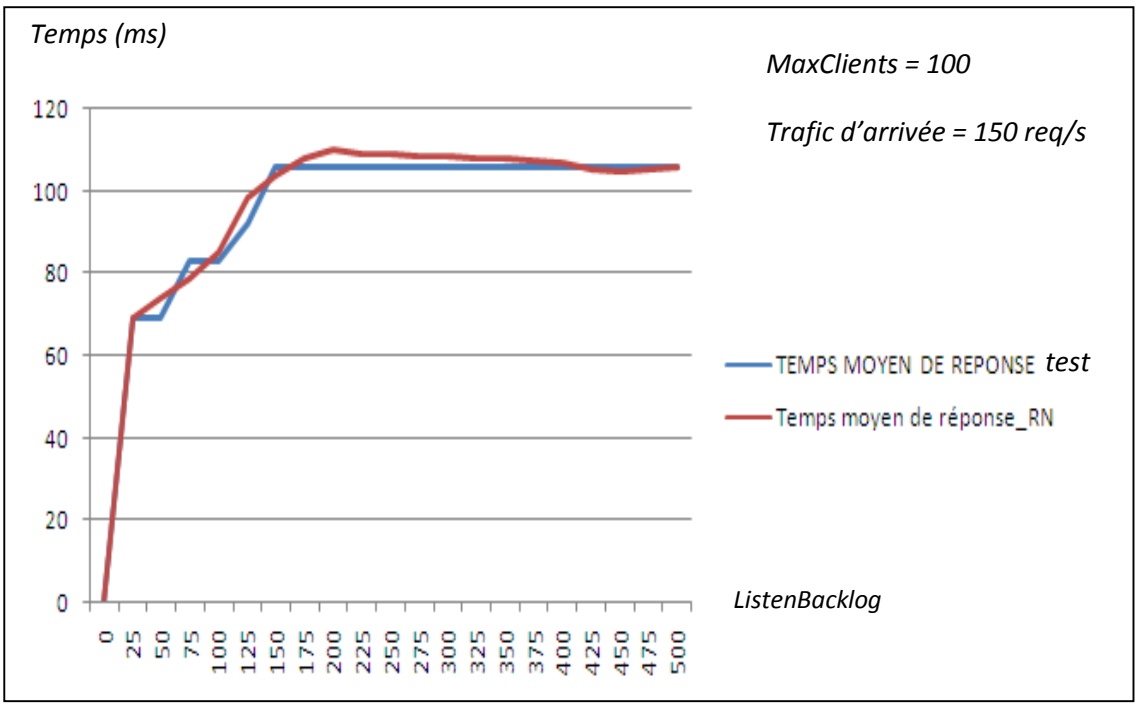


Figure IV.9 : Temps moyen de réponse par rapport au ListenBacklog

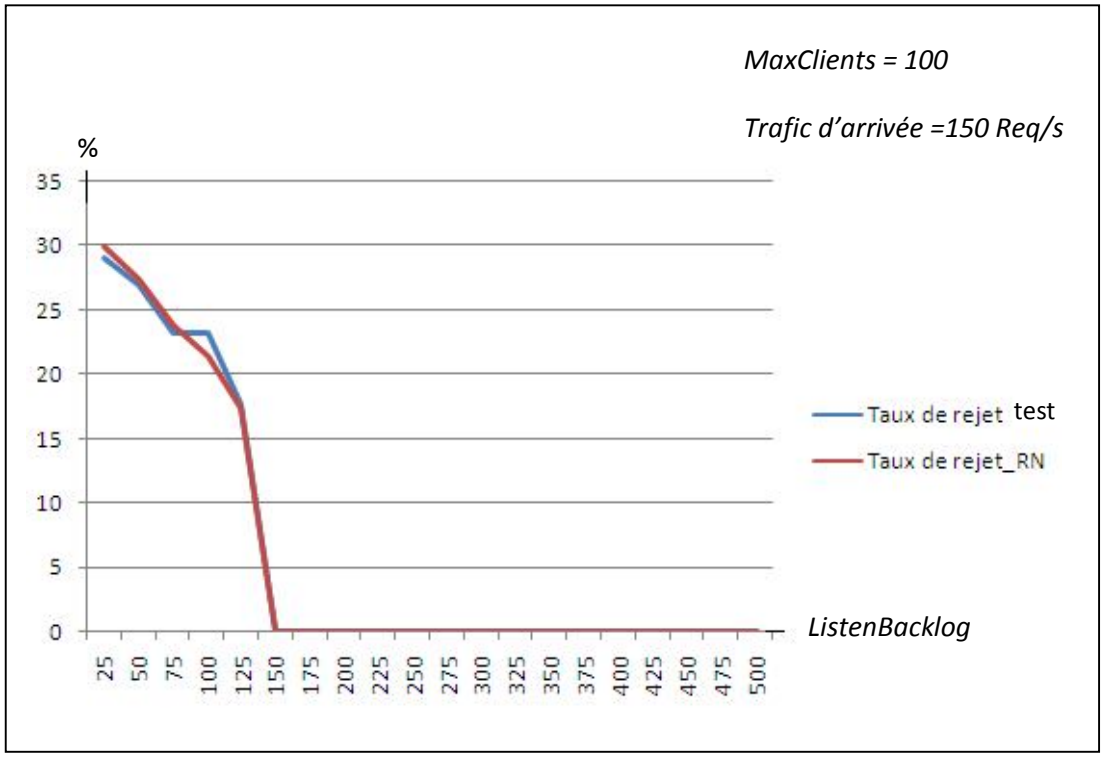


Figure IV.10 : Taux de rejet par rapport au ListenBacklog

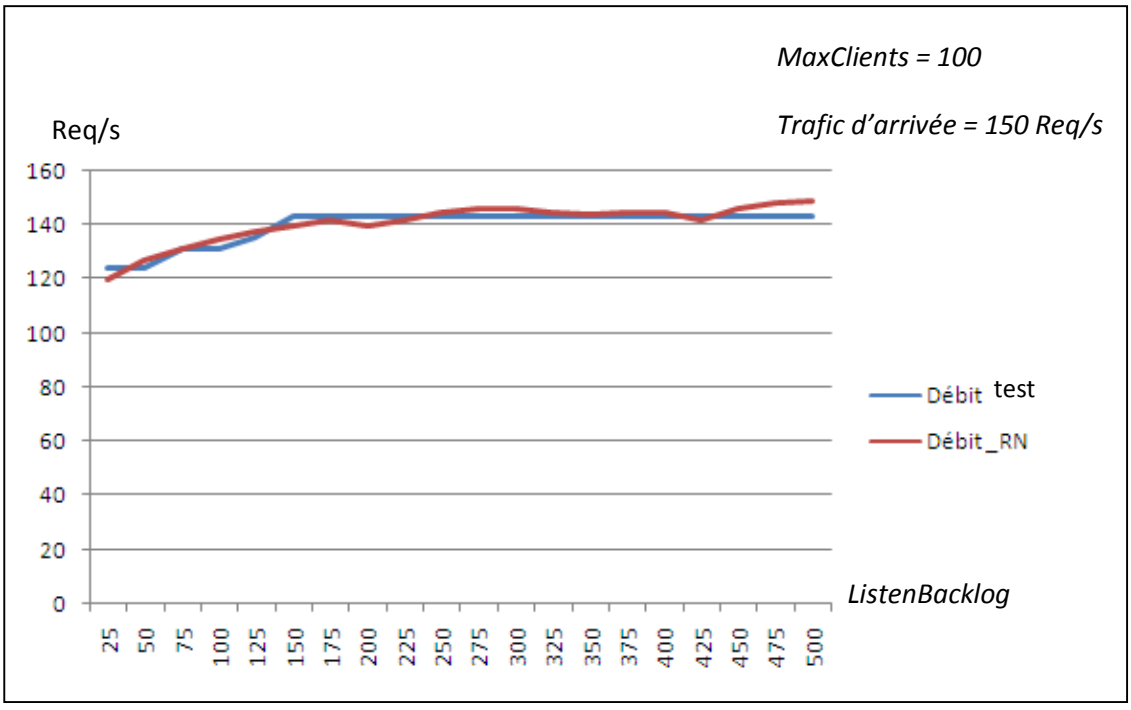


Figure IV.11 : Débit du serveur par rapport au ListenBacklog

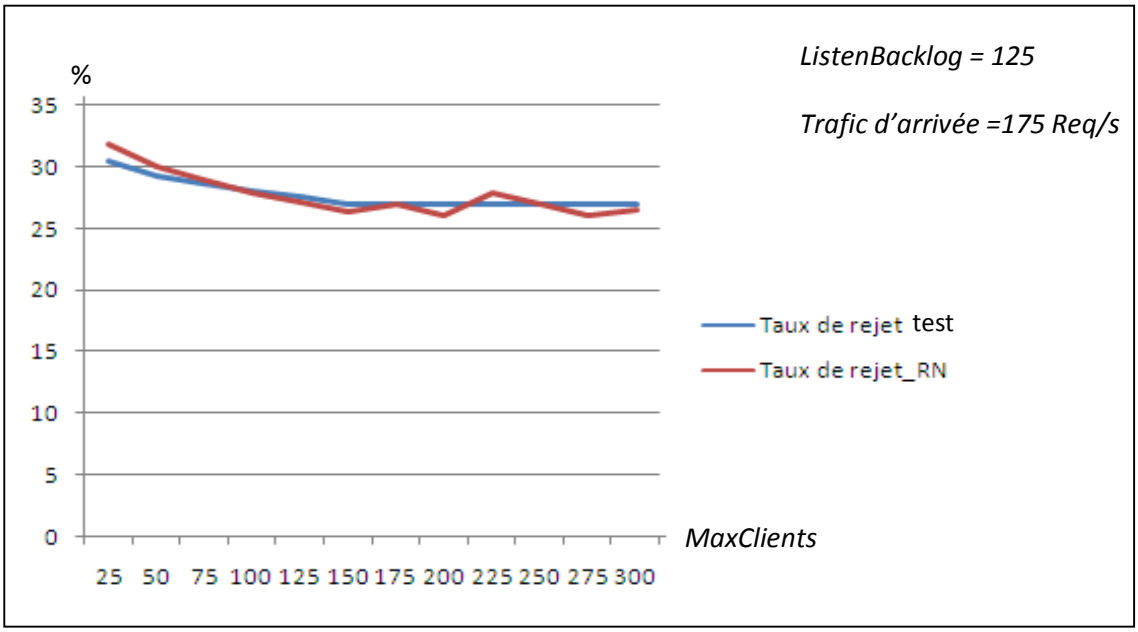


Figure IV.12 : Taux de rejet par rapport au MaxClients

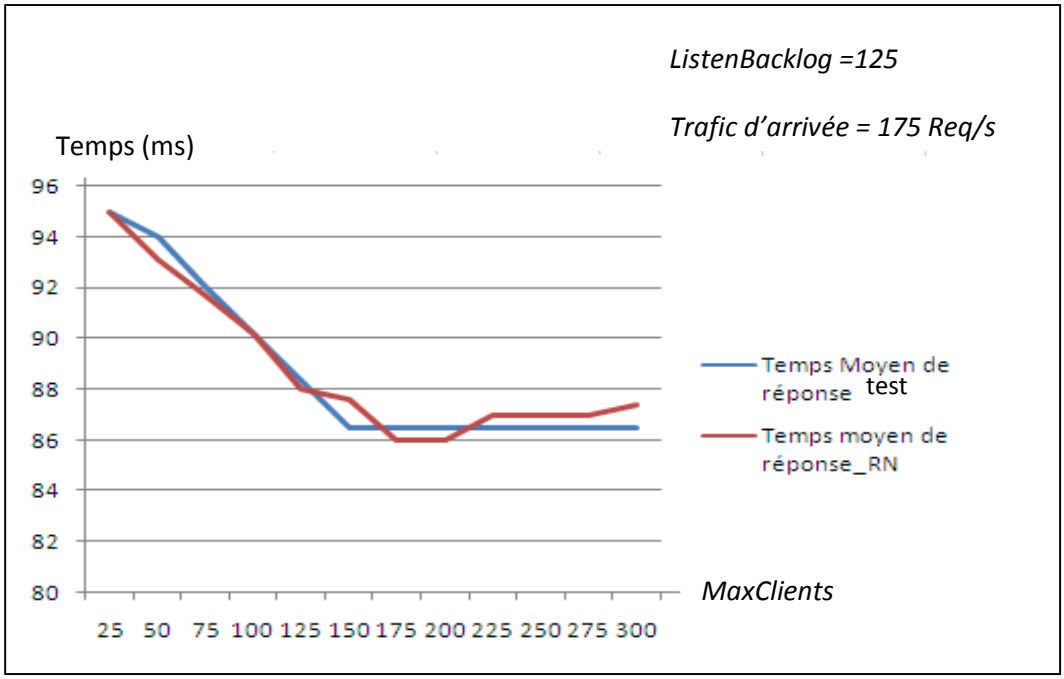


Figure IV.13 : Temps moyen de réponse par rapport au MaxClients

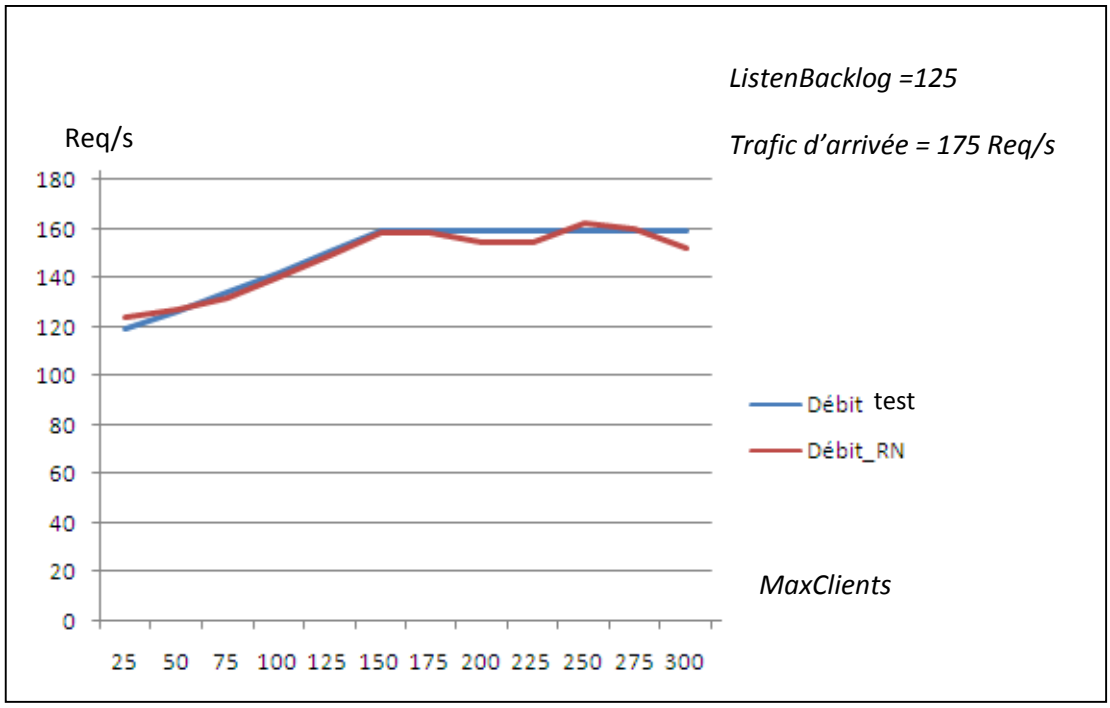


Figure IV.14 : Débit du serveur par rapport au MaxClients

IV.7 Implémentation

Nous avons reconverti le réseau de neurones obtenu de la phase d'apprentissage en langage C++ (plateforme Linux).

L'utilisateur n'a qu'a saisir les valeurs des paramètres (Maxclients, ListenBacklog , le trafic d'arrivée) , et le Réseau de neurone prédit instantanément les métriques de la Qds (Débit , Temps moyen de réponse, taux de rejet). (Figure IV.15).

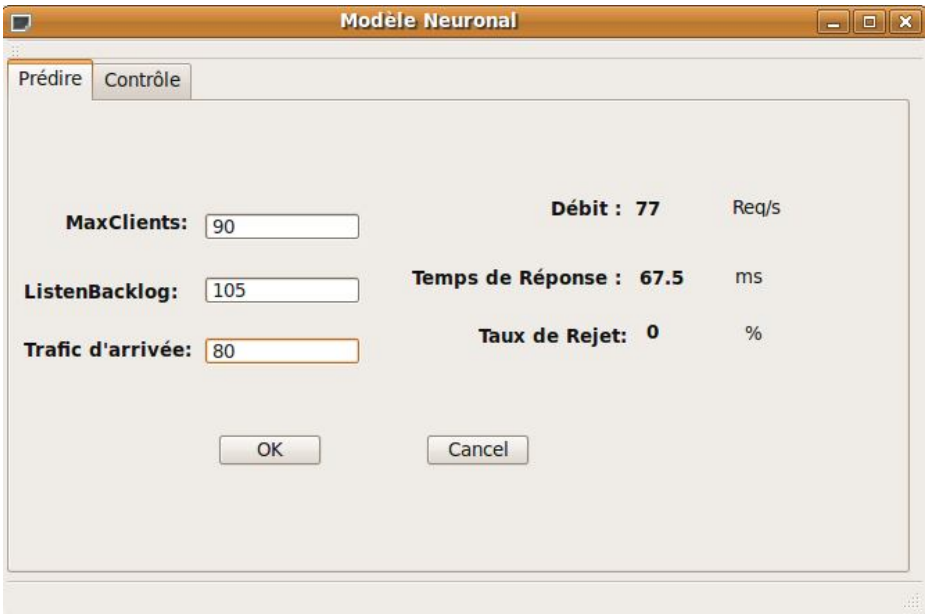


Figure IV.15 : interface du programme de prédiction

Après la réalisation de la prédiction, l'administrateur du serveur peut modifier les valeurs des paramètres : du serveur Apache MaxClients, ListenBacklog, Et du noyau de linux somaxconn (se trouvant dans le fichier /proc/sys/net/core/somaxconn)

En effet, après les avoir modifier dans le fichier de configuration d'apache (httpd.conf) , On effectue un *graceful restart*. Par cette technique, on redémarre apache avec les nouvelles valeurs sans perdre les connexions en cours (voir section II.2.6).

Le trafic d'arrivée sera limité grâce à *iptable* de Linux [PUR 04], en effet *iptable* est un module du noyau Linux réalisant le filtrage de paquets. Il possède plusieurs tables : *filter*, *nat* et *mangle*. Nous utiliserons ici la table *nat* qui contient les chaînes de traitement PREROUTING, POSTROUTING et OUTPUT, utilisées comme précisé par le schéma ci-dessous.

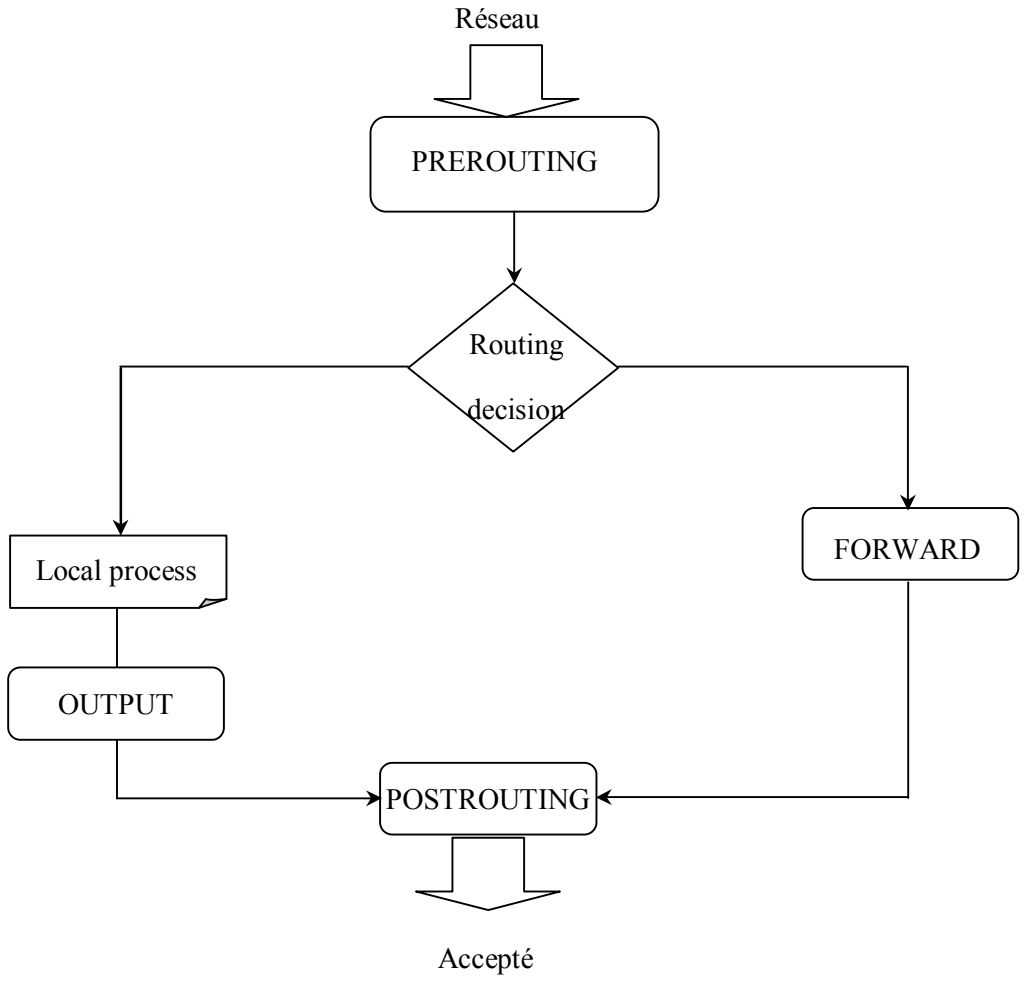


Figure IV.16 : Le trajet d'un paquet IP dans la table *nat* [PUR 04].

Un paquet arrivant par une interface réseau externe du serveur sera d'abord envoyé à la table *nat* de *iptables* pour être traité par la chaine PREROUTING. La chaîne est une liste de règles qui peut correspondre à un ensemble de paquets.

Nous allons créer une chaine qui permet de fixer une limite (*n*) de nombre de demandes de connexion au serveur.

iptables -t nat -N limiter_paquets

iptables -t nat -A limiter_paquets -m limit –limit <***n***>/s –limit-burst <***m***> -j RETURN

iptables -t nat -A limiter_paquets -j DROP

iptables -t nat -A PREROUTING -i $EXT_IFACE -d $DEST_IP -p tcp –syn -j limiter_paquets

Ainsi, le nombre des connexions TCP est limité à *n* connexion par seconde, après l'établissement de *m* connexion.

Cependant, lorsque l'administrateur a fixé les paramètres, il a effectué un contrôle d'admission basé sur une prédiction neuronale.

- **Rétroaction**

Si l'administrateur connait les valeurs des métriques de Qds (sorties du modèle) et veut savoir les valeurs des paramètres correspondantes (entrée du modèle), donc, par itérations (λ de 25 à 300, MaxClients et Listenbacklog de 25 à 500), on présente les entrées au Réseau de neurones, la sortie estimée est comparée à la sortie désirée. Ainsi on retient les entrées dont la sortie est plus proche ou égale la sortie désirée par l'administrateur (figure IV.17).

Cette technique est par fois lente mais donne des résultats exacts.

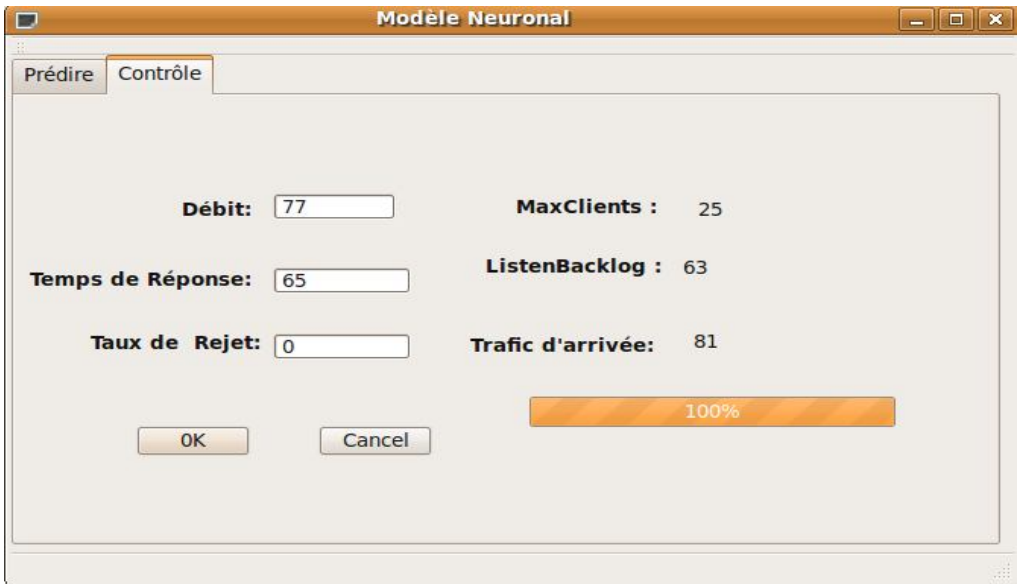


Figure IV.17 : interface du programme de contrôle

- **Surveillance du serveur web en temps réel**

Du moment que nous avons réalisé le réseau de neurones, nous pouvons alors surveiller le serveur web. En effet, si nous arrivons à compter le nombre des connexions par seconde il suffit de le présenter au réseau de neurone avec les deux autres paramètres du serveur Apache, afin de récupérer les valeurs des métriques de Qds.

Il faut compter le nombre de connexions TCP à destination du port 80 avant que le serveur web les accepte. Donc à un niveau plus bas.

Pour cela, nous avons décidé de réaliser un sniffer basé sur les sockets à bas niveau (*Linux raw sockets)* [LIN 10b].

- **Socket raw**

Une *raw socket* permet de lire des paquets, accéder à des protocoles de plus bas niveau comme IP, TCP.

Nous avons implémenté une fonction sniffer en C++ permettant d'écouter en permanence le trafic réseau et de filtrer les paquets ayant : une demande de connexion *syn* , le protocole *TCP*, le port de destination égal à 80. La fonction est lancée dans un thread à part.

La fonction compte le nombre de paquet filtré, chaque seconde (via un Timer) le compteur de paquet est remis à zéro. Ainsi, le compteur récupéré désigne le nombre de connexions par seconde.

Chaque seconde, le nombre des connexions est présenté avec les deux paramètres du serveur Apache au réseau de neurone, celui-ci estime les valeurs des métriques de Qds.

Donc, l'administrateur peut surveiller en temps réel le serveur web grâce aux valeurs des métriques de Qds fournies par le réseau de neurone.

En plus, lors de la surveillance, les métriques sont sauvegardées dans un fichier log, pour être exploitées à tout moment.

Nous avons implémenté le programme de surveillance d'une manière portable. L'administrateur à le choix, soit l'exécuter sur le serveur web, ou de le lancer sur une autre manche avant le serveur (figure IV.19). Les requêtes sont redirigées vers le serveur en utilisant *iptabes*.

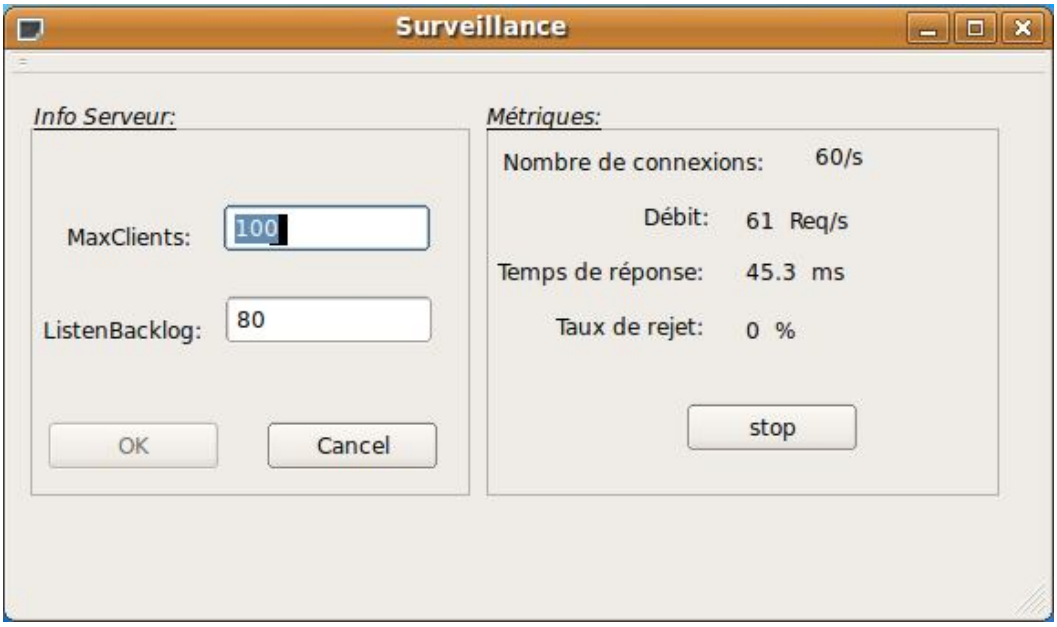


Figure IV.18 : interface du programme de surveillance

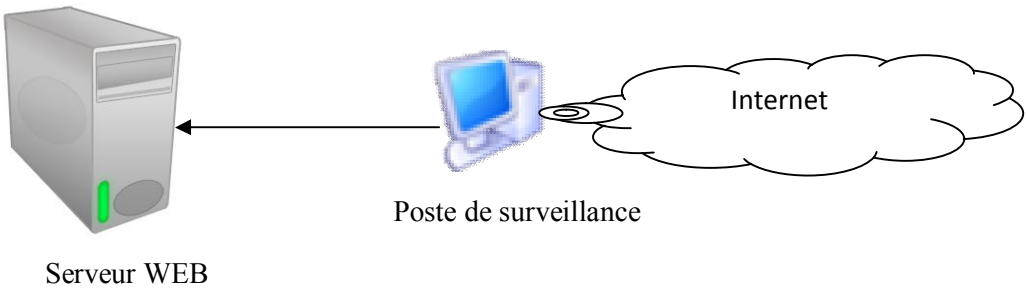


Figure IV.19 : surveillance du serveur via un autre poste

IV.8 Contrôle d'Admission basé sur la prédiction neuronale

Il est clair que le choix des paramètres d'un serveur a des conséquences directes sur sa performance, la disponibilité de son service et sa qualité de service. Les approches existantes de contrôle d'admission reposent sur un réglage ad-hoc, sur des heuristiques sans garantie d'optimalité (voir section I.11.1).

Même l'approche de [FON 07], dont le contrôle d'admission est réalisé à base du temps moyen de réponse estimé par un RNs. Cette approche ne prend pas en compte le pourcentage de rejet et le débit du serveur. Toutefois, si le contrôle d'admission est basé sur un seul critère de performance, les autres critères peuvent prendre des valeurs inconnues

même inacceptables par l'administrateur du serveur. Pour une valeur de temps de réponse il existe plusieurs valeurs pour le pourcentage de rejets et pour le débit.

Un administrateur d'un site e-commerce, par exemple, n'a pas d'intérêt d'avoir un temps de réponse raisonnable avec un taux de rejet très élevé.

Notre modèle prend en considération les trois métriques de la Qds à la fois pour effectuer un contrôle d'admission.

En effet, si un administrateur d'un site d'info veut publier une information (résultat des élections par exemple) en estimant le trafic d'arrivée, il peut donc prédire les valeurs des critères de Qds suivant lesquels il configure le serveur web.

IV.9 Conclusion

Dans ce chapitre, nous avons présenté un modèle de réseau de neurones *feedforward* permettant la prédiction des métriques de performance d'un serveur web. Les entrées du modèle sont deux paramètres du serveur web et le trafic d'arrivée, alors que les sorties sont le débit, le temps moyen de réponse et le pourcentage du rejet.

Nous avons construit la base d'apprentissage par simulations, qui a permis de mettre en évidence les capacités des modèles neuronaux à apprendre et à généraliser le comportement d'un système réel aussi bien à faible qu'à forte charge.

Les métriques des performances estimées à partir du réseau de neurones s'accordent parfaitement aux résultats expérimentaux. Aussi, nous avons utilisé ce modèle pour réaliser un contrôle d'admissions optimal, ainsi que la surveillance en temps réel des métriques des performances du serveur web.

Conclusion générale

Les objectifs de notre travail se sont articulés autour de l'idée suivante : utilisation des capacités d'apprentissage et de généralisation des réseaux de neurones pour construire un modèle permettant la prédiction des performances d'un serveur web.
L'avantage de l'utilisation de réseaux de neurones pour estimer les métriques de performance est que ceux-ci ne présupposent rien sur le système qu'ils modélisent ils, s'adaptent aux exemples qu'on leurs propose.

Nous avons en premier temps étudié le fonctionnement interne du serveur web Apache ainsi que l'amélioration de ces performances par une configuration optimale, ce qui nous a permis de choisir les paramètres d'entrée de notre modèle.

Dans un second temps, la modélisation du serveur web Apache par file d'attente a pu nous mettre en évidence les métriques de performance de celui-ci. Celles-ci ont été adoptées en tant que sorties de notre modèle neuronal.

Le modèle ainsi établi permet à la fois :

- de prédire les métriques de performance d'un serveur web
- d'avoir une configuration optimale de ce dernier

Ainsi, l'administrateur du serveur web n'a pas besoin d'effectuer un réglage ad-hoc des paramètres du serveur web.

La dernière étape de nos travaux a consisté à implémenter le modèle élaboré afin de réaliser un système de surveillance en temps réel des métriques des performances du serveur web.

Ce travail gagnerait énormément à être enrichi et étendu par une architecture permettant aussi bien l'ajustement en temps réel des paramètres du serveur web ainsi qu'un contrôle d'admission basé sur une prédiction neuronale. En effet, nous voudrions réaliser une conception mettant en œuvre un contrôleur agissant sur la base d'une commande rétroactive ("feedback control") avec un double objectif :

- ajustement en premier temps des entrées du modèle de réseau suivant les sorties désirées par l'administrateur,
- veiller à garder la qualité de service désirée en modifiant les paramètres du serveur web au besoin.

Annexes

Liste des serveurs web utilisés dans le monde

Server	**Developed by**	**cost**	**Open source**	**licence**
Apache HTTP Server	Apache Software Foundation	Free	Yes	Apache License
Internet Information Services (IIS)	Microsoft	Included with newer Windows products	No	proprietary
Nginx	Igor Sysoev	Free	Yes	BSD variant
GlassFish	Sun Microsystems	Free RTU, Support Subscription plans from Sun - Sun supported version comes with additional Enterprise Manager	Yes	CDDL and GPL and Sun support entitlement license
NCSA HTTPd	Robert McCool	Non-commercial use free, Commercial use with fee	Yes	Free for Non-Commercial Use
thttpd	Jef Poskanzer for ACME Laboratories	Free	Yes	BSD variant
TUX web server	Ingo Molnár	Free	Yes	GPL
Zeus Web Server	Zeus Technology	£1100+	No	proprietary

Codes de réponse http :

Code	Message	Description
10x	**Message d'information**	**Ces codes ne sont pas utilisés dans la version 1.0 du protocole**
20x	**Réussite**	**Ces codes indiquent le bon déroulement de la transaction**
200	OK	La requête a été accomplie correctement
201	CREATED	Elle suit une commande POST, elle indique la réussite, le corps du reste du document est sensé indiquer l'URL à laquelle le document nouvellement créé devrait se trouver.
202	ACCEPTED	La requête a été acceptée, mais la procédure qui suit n'a pas été accomplie
203	PARTIAL INFORMATION	Lorsque ce code est reçu en réponse à une commande GET, cela indique que la réponse n'est pas complète.
204	NO RESPONSE	Le serveur a reçu la requête mais il n'y a pas d'information à renvoyer
205	RESET CONTENT	Le serveur indique au navigateur de supprimer le contenu des champs d'un formulaire
206	PARTIAL CONTENT	Il s'agit d'une réponse à une requête comportant l'en-tête *range*. Le serveur doit indiquer l'en-tête *content-Range*
30x	**Redirection**	**Ces codes indiquent que la ressource n'est plus à l'emplacement indiqué**
301	MOVED	Les données demandées ont été transférées à une nouvelle adresse
302	FOUND	Les données demandées sont à une nouvelle URL, mais ont cependant peut-être été déplacées depuis...
303	METHOD	Cela implique que le client doit essayer une nouvelle adresse, en essayant de préférence une autre méthode que GET
304	NOT MODIFIED	Si le client a effectué une commande GET conditionnelle (en demandant si le document a été modifié depuis la dernière fois) et que le document n'a pas été modifié il renvoie ce code.
40x	**Erreur due au client**	**Ces codes indiquent que la requête est incorrecte**
400	BAD REQUEST	La syntaxe de la requête est mal formulée ou est impossible à satisfaire
401	UNAUTHORIZED	Le paramètre du message donne les spécifications des formes d'autorisation acceptables. Le client doit reformuler sa requête avec les bonnes données d'autorisation
402	PAYMENT REQUIRED	Le client doit reformuler sa demande avec les bonnes données de paiement
403	FORBIDDEN	L'accès à la ressource est tout simplement interdit
404	NOT FOUND	Le serveur n'a rien trouvé à l'adresse spécifiée.
50x	**Erreur due au serveur**	**Ces codes indiquent qu'il y a eu une erreur interne du serveur**
500	INTERNAL ERROR	Le serveur a rencontré une condition inattendue qui l'a empêché de

		donner suite à la demande (comme quoi il leur en arrive des trucs aux serveurs...)
501	NOT IMPLEMENTED	Le serveur ne supporte pas le service demandé (on ne peut pas tout savoir faire...)
502	BAD GATEWAY	Le serveur a reçu une réponse invalide de la part du serveur auquel il essayait d'accéder en agissant comme une passerelle ou un proxy
503	SERVICE UNAVAILABLE	Le serveur ne peut pas vous répondre à l'instant présent, car le trafic est trop dense (toutes les lignes de votre correspondant sont occupées veuillez rappeler ultérieurement)
504	GATEWAY TIMEOUT	La réponse du serveur a été trop longue vis-à-vis du temps pendant lequel la passerelle était préparée à l'attendre (le temps qui vous était imparti est maintenant écoulé...)

Classement des sites web les plus visités dans le monde

Google a publié en juillet 2010 [GOO 10] une liste du classement des 1000 sites web les plus visités au monde, alors que google.com ne figure pas sur la liste !, on ne retient que les 20 premiers.

Rank	Site	Category	Unique Visitors	Reach	Page Views	Has Advertising
1	facebook.com	Social Networks	540 000 000	34.9%	570 000 000 000	Yes
2	youtube.com	Online Video	490 000 000	31.6%	69 000 000 000	Yes
3	yahoo.com	Web Portals	450 000 000	29%	70 000 000 000	Yes
4	live.com	Search Engines	370 000 000	23.9%	36 000 000 000	Yes
5	wikipedia.org	Dictionaries & Encyclopedias	310 000 000	19.8%	7 000 000 000	No
6	msn.com	Web Portals	280 000 000	17.7%	12 000 000 000	Yes
7	microsoft.com	Software	210 000 000	13.6%	3 000 000 000	Yes
8	baidu.com	Search Engines	170 000 000	11.1%	27 000 000 000	Yes
9	qq.com	Email & Messaging	130 000 000	8.4%	19 000 000 000	Yes
10	mozilla.com	Internet Clients & Browsers	130 000 000	8.3%	2 000 000 000	Yes
11	adobe.com	Multimedia Software	120 000 000	7.5%	1 300 000 000	Yes
12	wordpress.com	Blogging Resources & Services	120 000 000	7.4%	1 200 000 000	Yes
13	sina.com.cn	Web Portals	110 000 000	6.9%	3 300 000 000	Yes
14	bing.com	Search Engines	110 000 000	6.8%	3 000 000 000	Yes
15	twitter.com	Email & Messaging	99 000 000	6.3%	5 900 000 000	No
16	ask.com	Search Engines	98 000 000	6.3%	1 900 000 000	Yes
17	amazon.com	Shopping	81 000 000	5.2%	3 600 000 000	Yes
18	taobao.com	Classifieds	80 000 000	5.1%	4 000 000 000	Yes
19	apple.com	Mac OS	80 000 000	5.1%	1 300 000 000	Yes
20	ebay.com	Auctions	74 000 000	4.7%	8 600 000 000	Yes

Analyse des dix sites web les plus visités dans le monde

Nous avons analysé la page d'accueil (la première page visité) de chaque site web des dix premier sites les plus visités dans le monde. Et ce, pour déterminer le nombre d'objets par page, celui-ci représente le nombre de requêtes nécessaire pour afficher la page.

1) Google.com

Moteur de recherche d'informations.

URL : http://www.google.com/index.html

Taille de la page d'accueil : 13017 octets.

Nombre d'objet : 03

Type d'objet	Nombre d'objets	Taille (octets)
HTML	1	4128
Images HTML	2	8889
Images CSS	0	0

2) Yahoo.com

Moteur de recherche d'informations

URL : http://www.yahoo.com/index.php

Taille de la page d'accueil : 134810 octets

Nombre d'objet : 45

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	36819
HTML Images:	23	74507
CSS Images:	16	16751
Javascript:	5	6733

3) Youtube.com

YouTube est un moyen d'obtenir et de partager des vidéos dans le monde entier.

URL : http://www.youtube.com/index.php

Taille de la page d'accueil : 112588 octets

Nombre d'objet : 09

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	2	12670
HTML Images:	6	24388
Javascript:	1	75530

4) Facebook.com

Facebook est un réseau social qui relie les gens, se tenir avec des amis, télécharger des photos, partager des liens et des vidéos.

URL : http://www.facebook.com/index.php

Taille de la page d'accueil : 95728 octets.

Nombre d'objet : 24

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	9352
HTML Images:	2	2731
CSS Images:	17	64626
Javascript:	2	10371
CSS imports	2	8648

5) Windows Live (live.com)

Moteur de recherche de Microsoft.

URL: http://login.live.com/index.asp

Taille de la page d'accueil : 14667 octets

Nombre d'objet : 12

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	3	4755
HTML Images:	2	1196
Javascript:	4	6658
CSS imports	3	2058

6) MSN.com

MSN : portail pour faire du shopping , des nouvelles , e- mail, recherche, et le chat.

URL: http://www.msn.com/index.asp

Taille de la page d'accueil : 151846 octets

Nombre d'objet : 32

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	21836
HTML Images:	14	66722
CSS Images:	11	37148
Javascript:	6	26140

7) Wikipedia.org

Wikipedia : une encyclopédie en ligne collaborative.

URL: http://www.wikipedia.org

Taille de la page d'accueille : 84618 octets

Nombre d'objet : 17

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	11568
HTML Images:	13	65603
CSS Images:	0	0
Javascript:	0	0
CSS:	3	7447

8) Blogger.com

Blogger : Outil de publication, gratuit, automatisé weblog qui envoie des mises à jour d'un site via FTP.

URL: http://www.blogger.com

Taille de la page d'accueil : 75714 octets

Nombre d'objet : 09

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	4719
HTML Images:	1	2066
CSS Images:	0	0
Javascript:	6	11719
CSS	1	57210

9) baidu.com

Baidu : est le premier moteur de recherche chinois(en langue chinoise).

URL: http://www.baidu.com/index.html

Taille de la page d'accueil : 13702 octets

Nombre d'objet : 04

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	2137
HTML Images:	2	1580
CSS Images:	0	0
Javascript:	1	9985

10) qq.com

qq est le système de messagerie instantanée propriétaire le plus utilisé en Chine.

URL : http://www.qq.com

Taille de la page d'accueil : 186376 octets

Nombre d'objet : 40

Type d'objet	Nombre d'objets	Taille (octets)
HTML:	1	49281
HTML Images:	29	90783
CSS Images:	3	15888
Javascript:	5	28538
CSS:	1	1266
Multimedia:	1	620

Bibliographie

[ACK 85] D. H. Ackley, G. E. Hinton, T. J. Sejnowski, "A learning algorithm for Boltzmann machines", Cognitive Science, Vol. 9, No. 1, pp. 147–169, 1985.

[AID 11a] L.AID, M.LOUDINI, W.K.HIDOUCI, ''An Admission Control Mechanism for Web Servers using Neural Network ''. International Journal of Computer Applications, Foundation of Computer Science, New York, USA. Volume 15– No.5, pp. 14-19, February 2011

[AID 11b] L.AID, M.LOUDINI, W.K.HIDOUCI," Modélisation et contrôle d'un serveur web par l'utilisation des techniques neuronales", accepté pour présentation orale à la conférence Doctoriales STIC'11, 20-21 Avril 2011, Tiaret, Algérie.

[ALE 10] Alexa the web information company, http://www.Alexa.com , juillet 2010.

[AND 03] M. Andersson, M. Kihl and A. Robertsson, "Modeling and Design of Admission Control Mechanisms for Web Servers using Non-linear Control Theory", In Proceedings of ITCOM'03, Orlando, USA, Sept. 2003.

[AND 04] M. Andersson, "Overload Control and Performance Evaluation of Web Servers", PhD thesis, Department of Electrical and Information Technology, Lund Institute of Technology, 2004.

[AND 05a] M. Andersson, "Introduction to Web Server Modeling and Control Research", Technical Report, Lund Institute of Technology, 2005.

[AND 05b] M. Andersson, A. Bengtsson, M. Höst, C. Nyberg, "Web Server Traffic in Crisis Conditions", In Proceedings of the Swedish National Computer Networking Workshop, SNCNW, 2005.

[APA 10] "Apache web server", http://www.apache.org, consulté le 13/03/2010.

[AUT 10] http://www.xenoclast.org/autobench

[BAT 88] D. M. Bates, "Nonlinear Regression Analysis and Its Applications", Edition John Wiley & Sons, 1988.

[BER 96] T. Berners-Lee, R. Fielding and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", RFC 1945, Network Working Group, 1996.

[BRA 98] P. Barford, M. Corvella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", Proceedings of the ACM SIGMETRICS'98, 1998, pp. 151-160.

[CAO 03] J. Cao, M. Andersson, C. Nyberg, M. Kihl, "Web Server Performance Modeling Using an M/G/1/K*PS Queue", (Extended version), In Proceedings of the 10th International Conference on Telecommunications, Feb. 2003, Papeete, Tahiti.

[COM 10] Complete Survey, http://www.complete.com , juillet 2010.

[CHE 90] D. L. Chester, "Why two hidden layers are better than one", In Proc. of the Int. Joint Conf Neural Net., Washington DC, 265–268, 1990.

[CHE 01] X. Chen, P. Mohapatra, H. Chen, "An Admission Control Scheme for Predictable Server Response Time for Web Accesses", In Proceedings of the International World Wide Web Conference (10), Hong-kong, 2001

[COM 98] D. Comer, "TCP/IP Architecture, Protocoles, Applications", 3ème edition, InterEditions, 1998.

[CYB 89] G. Cybenko, "Approximation by superposition of a sigmoid function", Math. of Contr, Signals & Syst., Vol. 2, pp. 303–314, 1989.

[DIA 02] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D.M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server", Network Operations and Management Symposium, 2002.

[DRE 02] G. Dreyfus, J.M. Martinez., M. Samuelides, M.B. Gordon, F. Badran, S. Thiria et L. Hérault, "Réseaux de neurones, méthodologie et applications", Eyrolles, (2002).

[DU 06] K.-L. Du and M. N. S. Swamy, "Neural Networks in a Softcomputing Framework", Springer-Verlag London Limited, 2006.

[DUC 99] W. Duch, N. Jankowski N (1999) "Survey of neural transfer functions". Neural Comput Surveys 2:163–212

[ELN 04] S. Elnikety, E. Nahum, J. Tracey, W. Zwaenepoel: "A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites". In Proceedings of the 13th International Conference on World Wide Web, 2004.

[FIE 99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T.Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616. Internet Engineering Task Force (1999).

[FLA 10] "Flash web server," http : //www.cs.princeton.edu/~vivek/flash/, consulté le 15/03/2010.

[FON 07] R. Fontaine, P. Laurencot, A. Aussem, "Apprentissage des performances, surveillance en temps réel et contrôle d'admission d'un serveur Web utilisant les techniques neuronales", 4th International Conference: Sciences of Electronic Technologies of Information and Telecommunications, March 25-29, 2007, Tunisie.

[FUN 89] K. Funahashi (1989) "On the approximate realization of continuous mappings by neural networks". Neural Netw 2(3):183–192

[GEM 92] S. Geman, E. Bienenstock, R. Doursat, "Neural networks and the bias/variance dilemma", Neural Comp. Vol. 4, No. 1, pp. 1–58, 1992.

[GOO 10] "Classement des 1000 sites web les plus visités dans le monde", http://www.google.com/adplanner, mars 2010.

[GRO 04] B. Grone, A. Knopfel, R. Kugel, O. Schmidt, "The Apache Modelling Project", Technical Report ISRN LUTFD2/TFRT--3239--SE, Hasso-Plattner-Institue for Software Systems Engineering, July 2004.

[HEL 04] J. L. Hellerstein, Y. Diao, S. Parekh & D. M. Tilbury, "Feedback Control of Computing Systems", , IEEE Press- John Wiley & Sons Inc., 2004.

[HES 52] M. R. Hestenes, E. Stiefel, "Methods of conjugate gradients for solving Linear systems". J. of Res of National Bureau of Standards–B 49:409–436, 1952.

[HOP 82] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities". Proc. Nat. Acad. Sci., 79:2554–2558, 1982.

[HOR 89] K. M. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators". Neural Networks, 2:359–366, 1989.

[IIS 10] "Microsoft internet information services," http://www.microsoft.com/, consulté le 15/03/2010.

[JAI 91] R. Jain. "The Art of Computer Systems Performance Analysis". John Wiley & Sons, 1991

[JAN 88] P. Janssen, P. Stoica, T. Soderstrom, P. Eykhoff, "Model structure selection for multivariable systems by cross-validation", Int. J. Contr., 47:1737–1758, 1988.

[KHT 10] "khttpd web server," http://www.fenrus.demon.nl/, consulté le 13/03/2010.

[KOB 78] H. Kobayashi, "Modelling and Analysis: An Introduction to System Performance Evaluation Methodology", The Systems Programming Series, Addison Wesley, Reading, MA, 1978.

[KOH 82] T. Kohonen, "Self-organized formation of topologically correct feature maps", Biol. Cybern., Vol. 43, pp. 59–69, 1982.

[LEV 44] K. Levenberg, "A method for the resolution of certain non-linear problems in least-squares", Quart. Appl. Math., Vol. II, No. 2, pp. 164-168, 1944.

[LIN 08] E. Lindegren, "preparing the Apache http server for feedback control application". Master thesis, Lund University, February 2008.

[LIN 10a] Linux manual page, http://linux.die.net/man/2/listen, consulté le 11/06/2010.

[LIN 10b] Linux manual page, http://linux.die.net/man/7/raw, consulté le 12/06/2010.

[LOO 97] C. Loosley, F. Douglas, and A. Mimo, "High-Performance Client/Server", JohnWiley & Sons, 1997

[MAH 05] Antoine Mahul, "Apprentissage de la qualité de service dans les réseaux multiservices : applications au routage optimal sous contraintes", thèse de doctorat, université blaise pascal, France, 2005.

[MAR 63] D. W. Marquardt, An algorithm for least-squares estimation of non-linear parameters, J. Soc. Indust. Appl. Math., Vol. 11, No. 2, pp. 431-44, june 1963.

[MAR 03] E. Marcus and H. Stern, "Blueprints for High Availability", Wiley, Sept. 2003.

[MAT 10] Site web officiel de Math Works 2, http://www.mathworks.com.

[MEI 01] R. D. Van Der Mei, R. Hariharan, and P. K. Reeser, "web server performance modeling", Telecomunication systems, vol. 16, no. 3,4, pp. 361-378, 2001

[MCC 43] W. S. McCulloch, W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Bull of Math Biophysics, Vol. 5, pp. 115–133, 1943.

[MIK 03] A. Mikael, C. Jianhua, K. Maria, N. Christian, "Performance modeling of an Apache web server with bursty arrival traffic", Proceedings of the International Conference on Internet Computing (IC'03), pp. 508-514, 23-26 juin 2003, Las Vegas, Nevada, USA.

[MIN 69] M. L. Minsky, S. Papert, "Perceptrons", MIT Press, Cambridge, MA, 1969.

[MOL 93] M. F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning", Neural Networks, Vol. 6, N° 4, pp. 525-533, 1993.

[MOS 98] D. Mosberger and T. Jin. "HTTPerf A Tool for Measuring Web Server Performance". HP Research Labs.December 1998, volume 26 issue 3, ACM Sigmetrics Performance Evaluation Review

[MOU 01] I. Mountain, "The Business Case for Disaster Recovery Planning : Calculating the Cost of Downtime", 2001, http://www.ironmountain.com/dataprotection/resources/CostOfDowntimeIrnMtn.pdf.

[NAS 08] North American Systems International Inc. "The True Cost of Downtime", 2008, http ://www.nasi.com/downtime_cost.php.

[NER 93] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, S. Marcos, "Neural networks and non-linear adaptive filtering: unifying concepts and new algorithms", Neural Computation, Vol. 5, pp. 165-197, 1993.

[NET 10] Netcraft Survey, disponible à l'adresse: http://www.necraft.com/survey/.

[PAL 06] N. Palluat, "Méthodologie de surveillance dynamique à l'aide des réseaux neuro-flous temporels", Thèse de doctorat, Université de Franche-Comté, France, 2006.

[PIN 87] F. J. Pineda, “Generalization of back-propagation to recurrent neural networks”, Physical Rev. Lett., N° 59, pp. 2229–2232, 1987.

[PLA 86] D. Plaut, S. Nowlan, G. E. Hinton, “Experiments on learning by back propagation”, Technical Report, Carnegie-Mellon University, 1986.

[PUR 04] G. N. Purdy, “Linux Iptables: Pocket Reference”, O’Reilly Media, Inc., USA 2004.

[REE 95] R. Reed, R. J. Marks II, S. Oh, “Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter”, IEEE Trans. on Neural Networks, Vol. 6, N° 3, pp. 529–538, 1995.

[RUM 86] D. E. Rumelhart, G. E. Hinton, R. J. Williams, “Learning internal representations by error propagation”. In: Rumelhart DE, McClelland JL (Eds) Parallel distributed processing: Explorations in the microstructure of cognition, 1: Foundation, 318–362. MIT Press, Cambridge, 1986.

[QIN 07] W. Qin, Q. Wang, “An LPV approximation for admission control of an internet web server: Identification and control”, Control Engineering Practice, No. 15, pp. 1457–1467, 2007.

[RAF 05] F. Rafamantanantsoa, P. Laurençot, A. Aussem, “Analyse, Modélisation et Contrôle en Temps Réel des Performances d'un Serveur Web”, Rapport de Recherche LIMOS/RR-05-06, 10 mars 2005.

[SAM 06] S. Samarasinghe, “Neural Networks for Applied Sciences and Engineering - From Fundamentals to Complex Pattern Recognition”, CRC Press, USA, 2006.

[SCU 01] V. -M. Scuturici, “Utilisation efficace des serveurs Web en tant que serveurs vidéo pour des applications de vidéo à la demande”, Thèse de doctorat, Université Lumière Lyon 2, France, 2001.

[STA 10] Statbrain Survey, http://www.statbrain.com.

[SUN 08] Q. Sun, G. Dai, W. Pan, “LPV Model and Its Application in Web Server Performance Control”, College of Automation Northwestern Polytechnical University Xi’an, China 2008 International Conference on Computer Science and Software Engineering.

[TUX 10] “Tux Web Server”, Reference manual : http://www.redhat.com/docs/manuals/tux/TUX-2.1-Manual/, consulté le 13/03/ 2010

[UIT 94] E.800, “Terms and definitions related to quality of service and network performance including dependability. Recommandation”, ITU (UIT: Union Internationale des Télécommunications), 1994.

[SPE 96] The Standard Performance Evaluation Corporation. SpecWeb96 at http://www.spechbench.org/osg/web96.

[WAI 00] P. Wainwright, “Apache professionnel”, Editions Eyrolles, 2000.

[WAI 04] P. Wainwright. “Pro Apache”, Third Edition. Apress, 2004.

[XIA 05] C. Xiang, S. Q. Ding, T. H. Lee, “Geometrical interpretation and architecture selection of MLP”, IEEE Trans. on Neural Networks, N° 16, Vol. 1, pp. 84–96, 2005.

[YAS 00] F. Yasuyuki, M. Masayuki, M. Hideo, “Performance modeling and evaluation of web server systems with proxy caching”, Ph.D thesis, Osaka University, 2000.

[YEA 96] N. J. Yeager et R. E. McGrath, “Web Server Technology: The Advanced Guide for World Wide Web Information Providers”, Morgan Kaufmann Publishers, 1996.

[YEU 99] K. H. Yeung, C. W. Szeto, “On the modeling of www request arrivals”, in Proceedings of the 1999 International Workshops on Parallel Processing, pp. 248–253, 1999.

[ZEU 10] “Zeus web server,” http://www.zeus.com, consulté le 13/03/2010.