Cours Evaluation de performances des Systèmes Informatiques

Mouad Ben Mamoun
Master Offshoring Informatique Appliquée
Département d'Informatique, Université Mohammed V-Agdal
email:ben_mamoun@fsr.ac.ma

Evaluation de performances

- *Objectif:* Calculer les indices de performances d'un système informatique. Par exemple: débit, temps de réponse, taux de perte, taux d'utilisation ...
- *Quand :* en phase de conception et d'exploitation pour :
  - Comparer différentes alternatives (architectures, algorithmes de routage, ...)
  - Dimensionner le système, càd, déterminer le nombre et la taille des composants pour répondre aux objectifs fixés
  - Améliorer les performances d'un système existant

Les systèmes étant coûteux, il faut prédire leurs performances avant de les construire ou de les modifier

Comment évaluer les performances d'un système?

Les techniques utilisées pour l'évaluation de performances sont :

- *Les méthodes analytiques :* on modélise le système grâce à un formalisme mathématique (réseaux de files d'attentes, chaînes de Markov, ...) et on calcule les indices de performances par des formules analytiques ou des méthodes numériques
- *La simulation :* on reproduit l'évolution du système par un programme informatique et on estime les indices de performances à partir des résultats observés
- *Les mesures :* sur le système réel ou sur des prototypes

Techniques d'évaluation de performances: avantages et inconvénients (1)

- *Les méthodes analytiques :*
  - peu de modèles qu'on sait résoudre simplement de façon exacte, complexité numérique élevée pour les modèles Markoviens
  + résultats sans incertitude, résultats paramétriques
  + les formules analytiques sont peu coûteuse en temps de calcul
- *La simulation :*
  - une simulation n'est qu'un déroulement possible, les résultats (estimations) sont fournis avec une incertitude statistique
  - les temps d'exécution peuvent être très importants
  + on peut simuler des modèles complexes, plus réalistes que ceux que l'on peut résoudre analytiquement

Techniques d'évaluation de performances: avantages et inconvénients (2)

- *Les mesures :*
  - \- technique très coûteuse et n'est pas toujours possible
  - \+ résultats plus crédibles
  - \+ peuvent servir comme trace pour des simulations ou pour valider des modèles analytiques

Remarque: il est intéressant d'utiliser parfois plusieurs techniques pour vérifier et valider les résultats

Terminologie

- *Variables d'état :* ensemble de variables décrivant l'état du système à un instant donné (ex. nombre de clients par file d'attente, nombre de connexions à un site Web, ...)
- *Espace d'état :* ensemble des valeurs que peuvent prendre les variables d'état
- *Événement :* un changement dans l'état du système (ex. arrivée ou départs de clients)
- *Système à événements discrets :* système à espace d'états discret
  (le temps peut être continu mais les événements occurrent de façon discontinu dans le temps à des instants aléatoires en général et provoquent des changements discrets des variables d'état)

Caractérisation stochastique des systèmes?

- En général, on ne connaît pas avec certitude les caractéristiques des systèmes étudiés, par exemple:
  - instants d'arrivées des connexions sur un site Web
  - durées des connexions,...

    $\Longrightarrow$ besoin de décrire l'aléatoire
- caractériser l'aléatoire est possible avec des distributions de probabilités, mais il faut tester l'adéquation à la réalité
- Dans ce cours, on se restreint aux modèles stochastiques pour des systèmes à événements discrets : modèles de systèmes informatiques et de réseaux

Contenu du module

- Bref rappel de probabilités et statistiques
- Simulation à événements discrets
- Chaînes de Markov
- Files d'attentes

Bibliographie

- Raj Jain, "The Art of Computer Systems Performance Analysis", John Wiley & Sons, 1991
- Bruno Baynat, "Théorie des Files d'Attente", Hermes-Lavoisier, 2000
- Georges Fiche, Gérard Hébuterne, "Trafic et performances des réseaux des télécoms", Hermes-Lavoisier, 2003
- William J. Stewart, "Introdution to the Numerical Solution of Markov Chains", Princeton University Press, 1994
- Bloch, Greiner, Meer, Trivedi, "Queueing Networks and Markov Chains, Modeling and Performance Evaluation with Computer Science Applications", John Wiley & Sons, 1998

Contenu de la partie simulation

- Introduction : structure d'un simulateur à événements discrets, gestion de l'échéancier, quelques outils de simulation, ...
- Initiation à l'outil de simulation QNAP
- Génération aléatoire: générateurs de nombres aléatoires, tests de générateurs, méthodes de génération selon une loi donnée, distributions utilisées en modélisation
- Analyse statistique des résultats de simulation : estimation de précision par intervalles de confiance, étude de la représentativité de la simulation, organisation des expériences

Rappel de probabilités : exercice

La fonction de densité de la loi exponentielle de paramètre $\lambda(\lambda > 0)$ est :

$$f_{\mathrm{EXP}(\lambda)}(x) = \lambda e^{-\lambda x}, \quad x > 0$$

1. Calculer la moyenne, la variance et la fonction de répartition de la loi exponentielle.
2. Montrer la propriété "sans mémoire" de la loi exponentielle :

$$Prob(X \leq x + t | X > x) = Prob(X \leq t)$$

Simulation à Evénements Discrets

- *Systèmes considérés (Systèmes à Evénements Discrets) :*
  - les événements $e_1, e_2, \ldots$ surviennent à des instants $t_1, t_2, \ldots$
  - l'état du système change de façon discrète aux instants t_i
  - l'état du système ne change pas entre deux événements consécutifs

    $\Longrightarrow$ il n'est pas utile de considérer les périodes inter-événements
- *Principe:* une simulation dirigée par les événements où on saute d'un événement au suivant et on note les changements d'états correspondants (mise à jour des variables d'états)
- *Réalisation:* nécessite un ensemble de tâches qui sont communes à tous les simulateurs à événements discrets indépendemment du système étudié

Tâches d'un simulateur à événements discrets

- *Gestion d'un échéancier d'événements* : maintien d'une liste ordonnée d'événements futurs en perpétuelle évolution
- *Maintien d'une horloge de simulation* : maintien d'une variable globale représentant le temps simulé
- *Génération de durées aléatoires* : pour simuler le temps écoulé entre deux événements (ex: entre deux arrivées successives)
- *Mises à jour des variables d'états à l'exécution de chaque événement*
- *Calculs statistiques* : pour quantifier les indices de performances (objectif de la simulation)
- *Trace de simulation* : pour débugger le programme de simulation

L'échéancier

- L'échéancier contient l'ensemble d'événements futurs. Chaque événement est représenté par : sa date d'occurrence, son type, un pointeur vers le code à exécuter à cette date
- Deux opérations fréquentes sur l'échéancier :
  - Insérer un nouveau événement
  - Chercher et ôter l'événement de plus petite date
- Le nombre d'événements futurs peut être très important à un instant donné $\Longrightarrow$ nécessité d'optimiser la gestion de l'échéancier
- Plusieurs structures de données possibles pour l'échéancier : tableau, liste chaînée simple ou double, Arbre, ...
- Le choix de la structure de données affecte le temps processeur nécessaire pour l'insertion et l'extraction et déterminera l'efficacité du simulateur

Algorithme général d'une simulation

1 Initialisations (du temps de simulation, des variables d'état et de l'échéancier avec un premier événement)

2 TantQue (Echéancier non vide et TempsSimul $<$ TMAX)

2.1 trouver l'événement Ev de plus petite date

2.2 TempsSimul= Ev.date

2.3 TraiterEvénement(Ev) (peut entraîner l'ajout ou la suppression d'événements de l'échéancier)

3 Fin de Simulation (Affichage des résultats statistiques)

Exemple élémentaire : simuler une file M/M/1/B

- File d'attente de taille B
- Inter-Arrivées exponentielles (moyenne λ)
- Services exponentielles (moyenne μ)
- On veut estimer par simulation la distribution du nombre de clients
- Critère de fin de simulation : t = tmax
  (t: temps courant de simulation, tmax : temps de fin de simulation)

Pseudo Code

```
Event    =    record
                date :   real;
                type :   (Arrivées, Service);
              end
```

Temps : un vecteur de réels de taille $B + 1$
AddEvent(date,type) : ajouter un événement
GetEvent(event) : extraire l'événement le plus proche
DelExp() : fonction de génération d'une durée aléatoire selon la loi Exponentielle

Pseudo Code

1 Temps:=0; etat:=0; t:=0;

2 dt:=DelExp(λ);

3 AddEvent(t+dt, Arrivées)

4 while $t < tmax$ do begin

4.1 GetEvent(e); ot:=t; t:=e.date;

4.2 Temps[etat]:=Temps[etat]+t-ot;

4.3 case **e.type** of

4.3.1 Service : begin

etat:=etat-1;

if etat> 0 then

AddEvent(t+DelExp(μ), Service);

end;

4.3.2 Arrivées : begin

if etat<B then

etat:=etat+1;

AddEvent(t+DelExp(λ), Arrivées);

if etat=1 then

AddEvent(t+DelExp(μ), Service);

end;

4.4 end;

Remarques :

- Une analyse de la dynamique montre qu'il y a un ou deux événements dans l'échéancier au cours de la simulation
- Inutile de faire une structure complexe dans ce cas
- Par contre, si on augmente le nombre de files, la taille de la structure augmente

Plan d'une étude par simulation

1. Définir un modèle pour le système
2. Ecrire le simulateur (implémenter le modèle)
3. Tester le simulateur
4. Choix des paramètres, Plan d'expériences
5. Analyse des résultats (éventuellement, recommencer en 4, en 2 ou en 1...)

Ecrire un simulateur

Plusieurs possibilités :

- Langage usuel : C, C++, Java, ...
  - \+ disponibilité, familiarité avec le langage, efficacité et flexibilité du code
  - \- temps de développement très long
- Outil de simulation spécialisé : NS, OPNET,QNAP, OMNeT, ...
  - \+ gain du temps dû à la gestion des tâches de la simulation à événements discrets, interfaces graphiques, lisibilité du code, plusieurs protocoles réseaux déjà codés dans certains outils
  - \- non disponibilité, certains outils sont payants, temps d'apprentissage de l'outil

Quelques outils de simulation

- **NS** (Network Simulator): logiciel libre, librairie riche de protocoles réseaux, apprentissage long, description de modèles à l'aide de scripts, C++ pour coder un nouveau protocole
- **OPNET**: logiciel commercial, librairie riche de protocoles réseaux (plus que NS), apprentissage long, description de modèles à l'aide d'une interface graphique, Proto-C pour coder (reconnaît C/C++)
- **QNAP** (Queueing Network Analysis Package): logiciel commercial, apprentissage facile, un langage de description de réseaux de files d'attentes, solveurs pour certains modèles mathématiques

Introduction à QNAP

QNAP est un outil comprenant :

- un langage de description de réseaux de files d'attente (mots clés pour décrire la durée de service, l'ordonnancement, la classe des clients, le routage,le nombre de clients, ...)
- un langage algorithmique (tests, boucles, ...) qui permet de décrire des services complexes
- un simulateur à événements discrets
- des solveurs analytiques (MVA, Diffusion, Forme Produit,...)

Éléments de syntaxe

- Un commentaire est compris entre un `&` et la fin de ligne
- Les chaînes de caractères sont de la forme `"abcd"`
- Les instructions se terminent par un point virgule `;`
- L'affectation se fait par un deux points égal `:=`
- Les majuscules et minuscules sont significatives
- Les mots réservés sont en majuscules (Ex: `ALL`, `AND`, `ANY`, `BEGIN`, `DO`, `ELSE`, `END`, `FALSE`, `FOR`, `IF`, `IN`, `IS`, `NIL`, `NOT`, `OBJECT`, `OR`, `REF`, `STEP`, `THEN`, `TRUE`, `UNTIL`, `WHILE`, `WITH`, ...)
- Pour les noms de variable, seuls les huit premiers caractères sont significatifs, les tabulations et autres caractères de contrôle ne sont pas autorisés

Structure d'un programme

```
/DECLARE/
        & Section de declaration des variables

/STATION/
        & Section de declaration de la premiere file d'attente
...

/STATION/
        & Section de declaration de la derniere file d'attente
/CONTROL/
        & Section de modification des valeurs par
        & defaut des parametres de controle

/EXEC/
        & Section contenant les instructions du programme

/END/
        & Fin du programme
```

Exemple de Programme

```
/DECLARE/
          QUEUE S, A;
/STATION/
          NAME = S;
          TYPE = SOURCE;
          SERVICE = EXP(1.0);
          TRANSIT = A;
/STATION/
          NAME = A;
          TYPE = SERVER;
          SERVICE = EXP(1.0);
          TRANSIT = OUT;
/CONTROL/
          TMAX = 1000;
/EXEC/
          SIMUL;
/END/
```

La section /DECLARE/

On y déclare des variables en donnant leur type et une éventuelle initialisation. Les types du langage sont :

- *Les types scalaires :* `INTEGER`, `REAL`, `BOOLEAN` et `STRING`
- *Le type tableau :* les indices du tableau se déclarent par `a:b` ou par simplement `a` qui est alors interprété comme `1..a`. Les éléments du tableau peuvent aussi être des files ou des clients
- *Le type objet:* pour les variables structurées. Il existe des types d'objets prédéfnis en QNAP, les principaux sont:
  - `QUEUE` : pour les files d'attente
  - `CUSTOMER` : pour les clients des files d'attente
  - `CLASS` : pour les classes de clients
  - `FLAG` : pour les synchronisations
- *Le type référence* `REF` : pour les pointeurs

Exemple de section /DECLARE/

```
/DECLARE/
    INTEGER I=2;
    BOOLEAN trouve;
    STRING nom;
    REAL Matrice (10,0:19);
    QUEUE A,B,C;
    QUEUE T(5);
    REF CUSTOMER F;
    FLAG feu;
    CLASS HP, BP;
```

Attributs

- Les types QUEUE et CUSTOMER ont des attributs prédéfinis
- L'accès à un des attributs se fait par un point. Par exemple `S.NB` est le nombre de clients dans la file `S`
- On peut ajouter de nouveaux attributs :

```
/DECLARE/
    CUSTOMER REAL InstArr;
    CUSTOMER INTEGER Numero;
```

Principaux attributs de QUEUE

- NB : nombre courant de clients dans la file
- NBIN : nombre total de clients entrés dans la file
- NBOUT: nombre total de clients sortis de la file
- FIRST: référence sur le premier client de la file
- LAST: référence sur le dernier client de la file

Principaux attributs de CUSTOMER

- PREVIOUS : référence sur le client précédent dans la file
- NEXT : référence sur le client suivant dans la file
- CQUEUE : référence vers la file contenant le client
- CPRIOR : niveau de priorité
- CCLASS : référence vers la classe
- LINK : référence sur le client suivant attendant sur le même flag
- SON: référence vers le dernier fils créé
- FATHER : référence vers le père (client créé par un NEW)

Le FLAG

- Un FLAG permet de bloquer un ensemble de clients et de les libérer tous en même temps pour effectuer une synchronisation
- Il a deux états: bloquant et passant
- `SET (f)` : Met le FLAG f dans l'état passant
- `RESET (f)` : Met le FLAG f dans l'état bloquant
- WAIT (f) : Un client qui rencontre cette instruction lors de son service doit attendre que le FLAG soit dans l'état passant

La section /STATION/

donne la description d'un objet de type QUEUE avec les paramètres : NAME, TYPE, SCHED, INIT, SERVICE, TRANSIT

- Le paramètre NAME est le nom de l'objet de type QUEUE déclaré. Il est obligatoire et doit être le premier de la section STATION.
- Le paramètre INIT permet d'initialiser le nombre de clients dans la file. Il est facultatif (par défaut la file est vide)
- Le paramètre TRANSIT indique la ou les files de destination pour le client en fin de service.

  Deux formulations possibles :

  TRANSIT = liste de stations et probabilités (la dernière peut être omise)

  TRANSIT = liste de stations et entiers (QNAP calcule des proportions)

Le paramètre `TYPE`

Il y a quatre types de base différents :

- `SOURCE` : file qui génère des clients
- `SERVER` : file au sens usuel
- `SEMAPHORE` et `RESOURCE` : c'est une file à jetons. Il n'y a pas de service
  - Un client n'est pas routé vers une file de ce type, mais peut faire `P(S)` pour demander un jeton à S ou `V(S)` pour libérer un jeton qui retourne dans S
  - L'action P(S) est bloquante si il n'y a pas de jeton. On attend le premier jeton disponible. Ces actions s'effectuent en général dans la phase de service d'une autre file
  - pour le type `RESOURCE` un client ne peut pas faire un V d'une ressource sur laquelle il n'a pas fait P au prealable

Ce paramètre est facultatif, par défaut une station est de type `SERVER`

Le paramètre `MULTIPLE (n)`

Pour les files de type :

- `SERVER`, `SEMAPHORE` , `RESOURCE`, on peut avoir plusieurs serveurs : `MULTIPLE (n)`
- `SERVER` , `SEMAPHORE`, on peut avoir une infinité de serveur : `INFINITY`
- Exemples :

```
TYPE = SERVER;                  & Un seul serveur
TYPE = SERVER, MULTIPLE(10);    & 10 serveurs
TYPE = SERVER, INFINITY;        & Une infinite de serveurs
```

Ce paramètre est facultatif, par défaut une station a un seul serveur

Le paramètre SCHED

Donne la discipline de service utilisée, par exemple :

- `FIFO` : First In First Out
- `LIFO` : Last In Last Out
- `PRIOR` : Les clients sont classés en fonction de leur priorité relative
- `PS` : Processor Sharing ; tous les clients sont servis simultanément avec une vitesse divisée par le nombre de clients. Cette discipline est incompatible avec la simulation

Ce paramètre est facultatif, par défaut la discipline de service est `FIFO`

Le paramètre `SERVICE`

- Si il est omis, les clients restent dans la file
- Si il y a un service mais pas de durée, le service est de durée nulle
- une séquence d'instructions encadrées par BEGIN et END ou une seule instruction de consommation de temps
- Instructions :
  - de consommation de temps (EXP, CST, HXP, ...)
  - de synchronisation (P, V, WAIT, SET, ...)
  - de mouvement de clients (TRANSIT)

Exemple de SERVICE

```
SERVICE =
  BEGIN
  EXP(10.0);                & La duree du service
  NbreC1 := NbreC1 + 1;     & Incrementation d'une variable globale
  V (TICKET) ;              & Liberation d'un semaphore
  F := NEW(CUSTOMER);       & Creation d'un nouveau client
  TRANSIT (F,B);            & Routage du nouveau client vers la file B
  IF CUSTOMER.CCLASS = C1 & Test de la classe du client en service
       THEN TRANSIT(A);     & Routage du client en service vers
       ELSE TRANSIT(B);     & la file A si sa classe est C1,
  END;                      & vers la file B sinon
```

La section /CONTROL/

permet d'initialiser certains paramètres de la méthode de résolution

Exemple:

```
/CONTROL/
  TMAX = Maxi;            & Duree de la simulation
  OPTION = TRACE;         & Affiche une trace du deroulement
                          & de la simulation
  ACCURACY = ALL QUEUE;& Calcul des intervalles de confiance
  MARGINAL = A,10;        & Permet d'obtenir les probabilites
                          & marginales de 1 a 10 clients dans A
  TSTART = 1000;          & Date a laquelle commencent
                          & les mesures statistiques
```

Sections /EXEC/ et /END/

- La section /EXEC/ contient l'appel de la méthode de résolution (ici SIMUL), et spécifie les instructions à exécuter avant et après cet appel (initialisations, affichages, ...). Exemple:

```
/EXEC/
  BEGIN
    Maxi := 10000;
    SIMUL;
    PRINT("Le nombre de paquets perdus est: ", NbrePertes)

  END;
```

- La section /END/ termine le programme Qnap, elle est obligatoire et ne contient que cette instruction

Résultats standards

```
 *** SIMULATION ***
... TIME =         10000.00
*************************************************************
*  NAME  *  SERVICE * BUSY PCT *  CUST NB * RESPONSE * SERV NB *
*************************************************************
*        *          *          *          *          *         *
* S      * .9985    * 1.000    * 1.000    * .9985    *    10015*
*        *          *          *          *          *         *
* A      * 1.015    * .9965    * 102.5    * 102.3    *     9818*
*        *          *          *          *          *         *
*************************************************************
... END OF SIMULATION ...
```

Dans ce tableau de résultats, les colonnes ont les significations suivantes :

- `NAME` : Nom de la file
- `SERVICE` : Temps moyen de service de la file
- `BUSY PCT` : Taux d'occupation de la file
- `CUST NB` : Nombre moyen de clients dans la file
- `RESPONSE` : Temps moyen de réponse de la file
- `SERV NB` : Nombre de clients ayant quitté la file

Générateurs de nombres aléatoires : motivation

- Une étape fondamentale dans le développement d'un simulateur est d'avoir des procédures pour générer des valeurs aléatoires selon une loi de probabilité donnée (ex: exponentielle ou normale)

- Ceci se fait en deux étapes :
  1. générer un nombre aléatoire uniformément distribué entre 0 et 1 (*random-number generation)*
  2. transformer ce nombre pour fournir une valeur selon la loi de probabilité désirée (*random-variate generation)*

Méthode de génération de nombres aléatoires

Dispositif physique ou Algorithme?

- *Contrainte* : on veut une génération rapide, des séquences exactement reproductibles (par exemple pour debugger, ...)

  $\Longrightarrow$ la méthode couramment utilisée est un algorithme

- *Paradoxe* : construction déterministe de l'aléatoire
  (on parle en général de nombres Pseudo-aléatoires)

- *Solution* : vérifier les propriétés aléatoires du générateur par des tests statistiques

Principe de Base

Utiliser une fonction f à n arguments pour générer la prochaine valeur. Les arguments sont les n valeurs précédentes de la série

$$x_{n+k} = f(x_{n+k-1}, x_{n+k-2}, \ldots, x_k)$$

- Il faut n valeurs pour initialiser le processus (racines, graines, semences, seed)
- Comportement cyclique potentiel. Si on retombe sur une série de n valeurs déjà calculées, alors la suite de la série est la même
- La longueur du cycle dépend de la fonction mais aussi des racines

Propriétés désirées pour les séquences générées

- avoir la plus grande longueur de cycle (période) possible
- réussir un test d'adéquation à la distribution uniforme
- réussir des tests variés :
  - distributivité dans l'espace
  - indépendance entre valeurs successives
  - ...

Choix de la fonction f

- La génération de nombres uniformes est une des fonctions les plus utilisées dans un programme de simulation

  $\Longrightarrow$ Attention à l'efficacité...

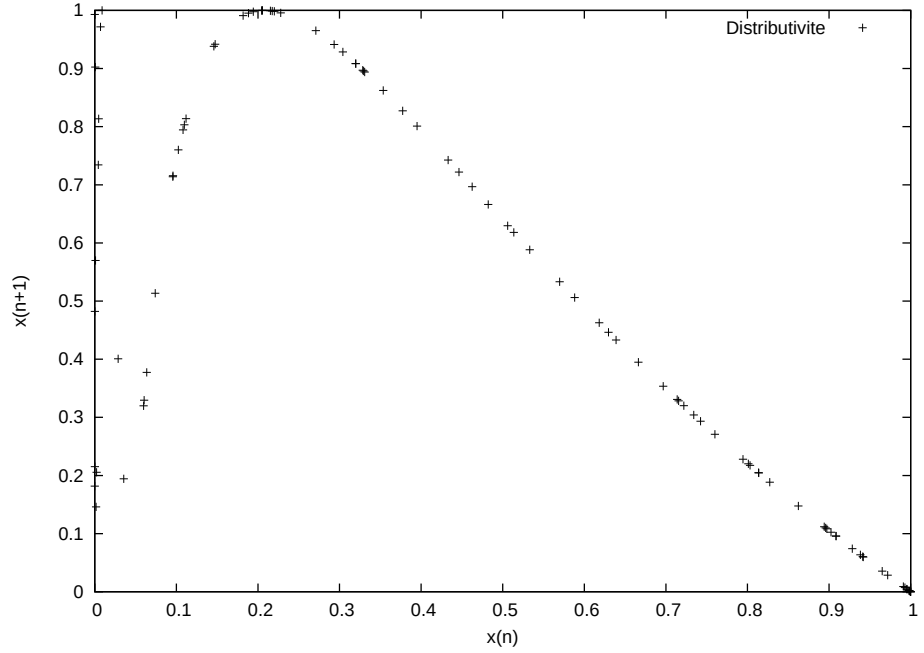
- Employer des opérateurs simples et rapides (arithmétique entière, décalage binaire)
- Compromis entre la complexité et des propriétés statistiques à vérifier
- Remarque: Ce n'est pas parce que ça a l'air compliqué que c'est aléatoire....

Exemple : mauvais générateur

$$x_{n+1} = abs(sin(log(x_n)))$$

Complexité : les fonctions *Log* et *Sin* sont beaucoup plus longs à calculer que des opérateurs arithmétiques de base (+, modulo, ...)

Distributivité : (100 tirages, $x_0 = 2$)



Générateur LCG

Un des générateurs les plus utilisées est le générateur LCG (Linear Congruential Generator)

- La fonction f est linéaire et utilise une arithmétique modulo m :

$$x_n = ax_{n-1} + b \ \ mod \ \ m$$

  - a et b sont positifs ou nuls
  - x_n est un entier entre 0 et $m - 1$
  - On se ramène aux réels par division par m :

$$y_n = x_n/m$$

    donc $y_n \in [0, 1[$

Propriétés des générateurs LCG

- Le choix de a, b, m, et x_0 affectent les propriétés du générateur (période, corrélation, ...)
- La période maximale est m
- Le calcul du modulo est plus efficace pour $m = 2^k$ (simple décalage à droite de k bits)
- Pour étudier les propriétés des LCG, on distingue les cas :
  - $b \neq 0$: dans ce cas on parle de LCG mixte
  - $b = 0$: dans ce cas on parle de LCG multiplicatif

LCG mixte ($b \neq 0$)

la période maximale m est obtenue si et seulement si les trois conditions suivantes sont satisfaites :

- b et m sont premiers entre eux
- chaque entier premier diviseur de m est aussi diviseur de $a - 1$
- si m est multiple de 4, $a - 1$ doit être multiple de 4

Exemple : les générateurs de type $m = 2^k$, $a = 4c + 1$, et b impair ont un cycle de longueur m

LCG multiplicatif

Deux familles intéressantes :

1. $m = 2^k$
2. m premier

$$m = 2^k$$

- L'opération modulo est un simple décalage binaire
- La période maximale est seulement $m/4$
- La période maximale est atteinte pour a de la forme $8i \pm 3$ (i est un entier positif ou nul) et pour une racine x_0 impaire

m premier

- Quand m est premier et $x_0 \neq 0$, tous les x_n sont différents de 0
- La période maximale est $m - 1$
- La période maximale est atteinte, si et seulement si, a est une racine primitive de m

Définition : a est une racine primitive de m, si et seulement si, $a^n \ mod \ m \neq 1$ pour $n = 1, 2, \ldots, m - 2$

Exemple: 3 est une racine primitive de 31, $x_n = 3x_{n-1} \ mod \ 31$ a une période maximale.

5 n'est pas une racine primitive de 31 ($5^3 \ mod \ 31 = 1$)

Problèmes d'implémentation des générateurs LCG

1. Il faut toujours travailler sur de l'arithmétique entière exacte et non pas passer en réel et faire une troncature
2. Vérifier que le type entier existe effectivement dans le langage et qu'il y a une arithmétique exacte associée
3. il faut que l'opération de multiplication $a \times x_{n-1}$ ne provoque pas d'erreurs de débordement (dépassement de l'entier maximum).

Sinon, les propriétés du générateur ne sont pas garanties

Méthode de Schrage

Pour éviter le problème du dépassement de l'entier maximal.

$$a \times x \ \ mod \ \ m = g(x) + m \times h(x)$$

$$\begin{aligned} q &= (m \ \ div \ \ a) && (1) \\ r &= (m \ \ mod \ \ a) && (2) \\ g(x) &= a \times (x \ \ mod \ \ q) - r \times (x \ \ div \ \ q) && (3) \\ h(x) &= (x \ \ div \ \ q) - (a \times x \ \ div \ \ m) && (4) \end{aligned}$$

Pour tout x de 0 à $m-1$, les termes apparaissant dans le calcul de $g(x)$ sont plus petits que m.

De plus, si $r < q$, $h(x)$ vaut 1 si $g(x)$ est négatif et 0 si $g(x)$ est positif.

La méthode de Schrage ne peut pas être utilisée si la condition $r < q$ n'est pas satisfaite.

Méthode de Schrage : exemple

Considérons le générateur

$$x_n = 7^5 \times x_{n-1} \ \ mod \ \ (2^{31} - 1)$$

$a \times x$ peut atteindre à peu près 2^{45}

Débordement possible sur une arithmétique 32 bits.

Méthode de Schrage:

$$a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647$$

On calcule q et r :

$$q = 12773, \quad r = 2836$$

$r < q$ et on peut éviter le calcul de $a \times x$

Quelques conseils

- Utiliser un algorithme récent étudié par un spécialiste
- Ne jamais utiliser 0 comme racine, si le générateur est un LCG multiplicatif, on reste coincé à 0
- Ne jamais prendre de racines aléatoires. Deux risques : non reproductibilité de la séquence et choix d'une mauvaise racine

  Exemple : pour le générateur $x_n = 9806x_{n-1} \ \ mod \ \ (2^{17} - 1)$, si par hasard on choisit $x_0 = 37911$, on reste coincé à cette valeur

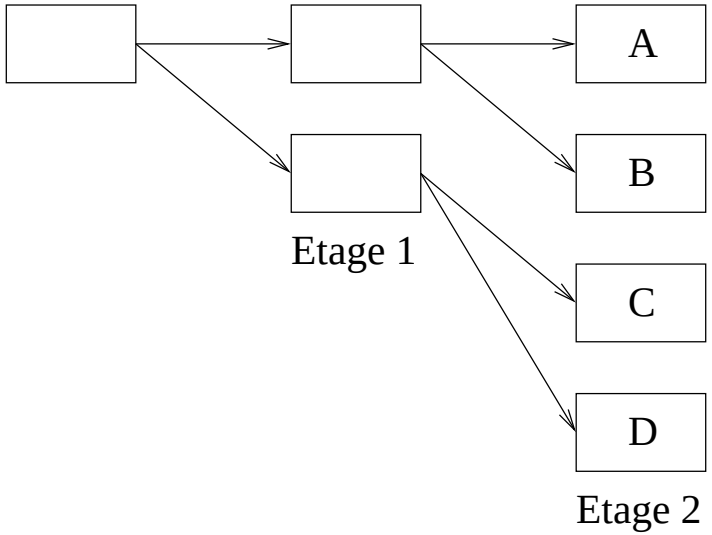
Ne jamais supposer que si un nombre est aléatoire, un bit de sa représentation l'est également

Exemple, $x_n = (8789x_{n-1} + 16384) \ \ mod \ \ 2^{16}$ initialisé à 1

- le bit 1 (le poids faible) est toujours 1
- le bit 2 est toujours 0
- le bit 3 suit la séquence 01
- le bit 4 suit la séquence 0110
- le bit 5 suit un cycle 11010010
- plus généralement le bit l suit un cycle de longueur 2^{l-2}.

Les bits de poids faible sont "peu aléatoires".
Le comportement cyclique des bits est toujours vrai pour les générateurs LCG avec $m = 2^k$.

Exemple d'erreurs provoquées par l'utilisation des bits de poids faible de ce générateur



On considère un réseau multi-étages où à chaque switch on choisit la destination avec probabilité $1/2 \Longrightarrow$ chaque switch du 2ème étage traite en moyenne $1/4$ du trafic

Si on utilise le bit 1 (resp 2) des entiers générés pour faire le routage vers le 1er (resp. 2ème) étage : 0 pour sortie haute, 1 pour sortie basse, alors le switch C traite la totalité du trafic

Exercices

Exercice 1

1. Quelle est la période du générateur : $x_n = (2^{18} + 1)x_{n-1} + 1 \bmod 2^{35}$
2. Quelle est la période maximale pour le générateur: $x_n = ax_{n-1} \bmod 2^4$. Quel doit être la valeur de a pour atteindre ce maximum. Y-a-t-il des restrictions sur la valeur de la racine ?

Exercice 2 Implémenter chacun des deux générateurs suivants avec et sans la méthode de Schrage :$x_n = 5x_{n-1} \mod 31$, $x_n = 11x_{n-1} \mod 31$ ($x_0 = 1$). est ce que les séquences générées sont les mêmes? sinon, expliquez pourquoi.

Exercice 3 Générer 48 nombres successifs à partir de la racine $x_0 = 1$ pour le générateur : $x_n = (13x_{n-1} + 11) \mod 2^{16}$
Trouvez la période du ième bit pour i de 1 à 5.

Générer selon une distribution quelconque

A partir des générateurs de réels uniformes entre 0 et 1,

5 méthodes complémentaires :

- transformation inverse
- rejet
- composition
- convolution
- caractérisation statistique

Transformation Inverse

Soit X une variable aléatoire et F sa fonction de répartition

$U = F(X)$ est uniformément distribué entre 0 et 1

Il suffit donc de générer X selon $F^{-1}(U)$, càd, on génére u uniforme entre 0 et 1 et on retourne $F^{-1}(u)$

Contrainte: on peut calculer facilement F^{-1} grâce à une formule analytique ou une distribution empirique

Exemple pour une formule analytique

La distribution exponentielle a pour fonction de répartition

$$F(x) = 1 - exp(-\lambda x)$$

Calcul de l'inverse :

$$F^{-1}(u) = \frac{-1}{\lambda} ln(1-u)$$

Remarque: si u est uniforme sur $[0,1]$ alors $(1-u)$ est également uniforme sur $[0,1]$

Générer selon la distribution exponentielle consiste à générer u uniforme entre 0 et 1 et retourner

$$x = \frac{-ln(u)}{\lambda}$$

Exemple pour une distribution empirique

On mesure les tailles de paquets sur un réseau et on obtient la distribution expérimentale suivante :

$$\begin{cases} 256 \ \textit{octets} : & 0.4 \\ 512 \ \textit{octets} : & 0.25 \\ 1024 \ \textit{octets} : & 0.35 \end{cases}$$

On calcule la fonction de répartition puis son inverse :

Fonction de Répartition

$$F(x) = \begin{cases} 0 & 0 \leq x < 256 \\ 0.4 & 256 \leq x < 512 \\ 0.65 & 512 \leq x < 1024 \\ 1.0 & 1024 \leq x \end{cases}$$

Inverse de la Fonction de Répartition

$$F^{-1}(u) = \begin{cases} 256 & 0 < u \leq 0.4 \\ 512 & 0.4 < u \leq 0.65 \\ 1024 & 0.65 < u \leq 1 \end{cases}$$

Attention à l'éfficacité : ordonner les tests par probabilité décroissante

Exemple : transformation inverse pour la loi de Poisson

$$Prob(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}$$

Pas d'expression simple pour la fonction de répartition et son inverse

$F(n) = \sum_{k=0}^{n} Prob(X = k), \quad Prob(X = k) = \frac{\lambda}{k} Prob(X = k - 1)$

Calcul de l'inverse par un algorithme :

```
P = exp (-lambda); X=0;
U=Random; F=P;
while (U > F) {
      X=X+1;
      P=P*lambda/X;
      F=F+P;
      }
return X;
```

Tabulation de la tête de la distribution qui est souvent calculée.

exemple : tableau de taille 10 pour lambda=1

i	0	1	2	3	4
F_i	$0,36787944$	$0,73575888$	$0,91969861$	$0,98101184$	$0,99634015$
i	5	6	7	8	9
F_i	$0,99940581$	$0,99991675$	$0,99998975$	$0,99999887$	$0,99999989$

Algorithme efficace : une recherche sur les 10 valeurs tabulées suivi de l'algorithme général pour $i > 9$.

Conclusion : on utilisera seulement les 10 valeurs tabulées, sauf dans 11 cas sur 100 millions

max, maxF, maxP et F ont été initialisés...

```
U=Random;
if (U <= maxF) {
      X=0;
      while (U>F(X)) X=X+1;
      }
else {
      X=max; F=maxF; P=maxP;
      while (U>F) {
            X=X+1;
            P=P*lambda/X;
            F=F+P;
          }
      }
return X;
```

Rejet

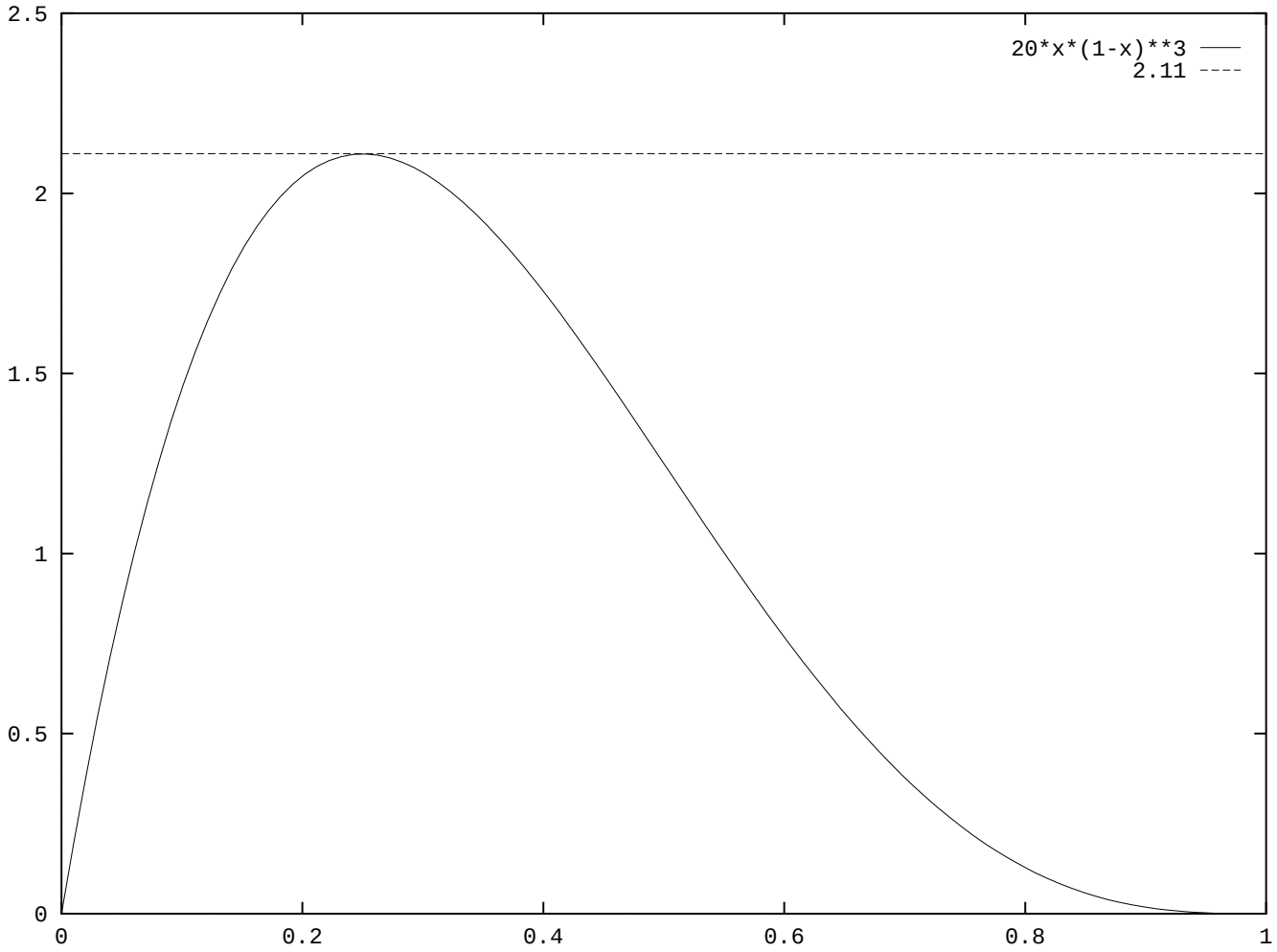
Hypothèse : il existe une densité $g(x)$ et une constante c telle que $f(x) \leq c\ g(x)$ pour toute valeur de x.

La méthode consiste alors à :

1. générer x de densité $g()$,
2. générer y uniforme sur $[0, cg(x)]$,
3. si $y \leq f(x)$ rendre la valeur x, sinon revenir en 1.

Il a été prouvé que ce procédé permet de générer une variable aléatoire distribuée selon la densité f

Exemple : méthode du rejet



Remarques sur la méthode de rejet

- g(x) doit être une densité (g(x) est positive et $\int g(x)dx = 1$)
- $g(x)$ doit être facile à générer
- $cg(x)$ doit être très proche de $f(x)$. Sinon, il faudra de nombreuses tentatives avant que le test soit positif. Et la complexité de la génération sera trop élevée.

Exemple : Méthode du Rejet

Exemple : Générer selon une loi Beta(2,4) dont la densité est $f(x) = 20x(1-x)^3$ avec $x \in [0,1]$.

Lorsque x est compris entre 0 et 1, la fonction $f(x)$ est plus petite que 2.11.

Choix possible : $c = 2.11$ et $g(x) = 1$ (g : uniforme sur $[0,1]$).

1. générer x uniforme sur $[0,1]$,
2. générer y uniforme sur $[0, 2.11]$,
3. si $y \leq 20x(1-x)^3$ retourner la valeur de x, sinon revenir en 1.

Composition

Hypothèse : la fonction de répartition (ou la fonction de densité) est la somme pondérée d'autres fonctions de répartition (ou fonctions de densité):

$$F(x) = \sum_{i=1}^{n} p_i F_i(x) \quad (f(x) = \sum_{i=1}^{n} p_i f_i(x))$$

avec $p_i \geq 0$ pour tout i, et $\sum_{i=1}^{n} p_i = 1$.

La méthode consiste à

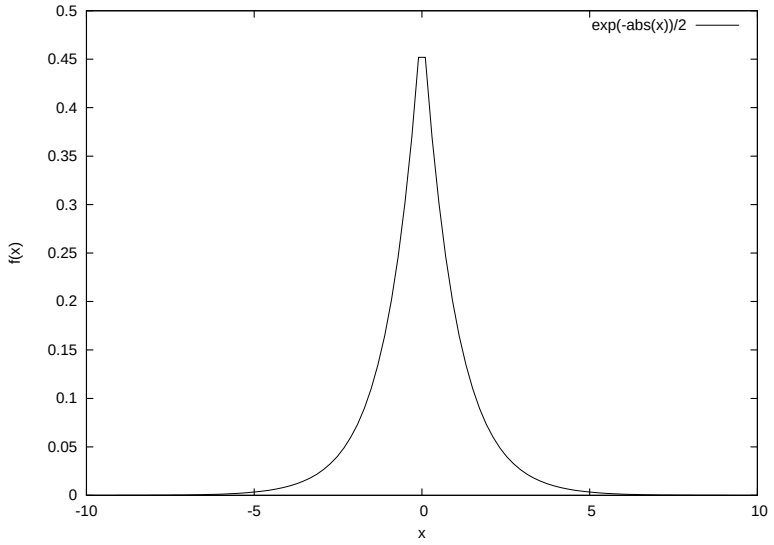
1. générer un entier k selon la distribution $(p_1, p_2, \ldots, p_n)$,
2. générer x selon la $k$ème fonction de répartition F_k

Exemple : Méthode de Composition

La densité de la loi de Laplace définie sur $\mathcal{R}$ est : $f(x) = \frac{1}{2a} e^{-|x|/a}$

Cette densité peut s'écrire: $f(x) = \frac{1}{2} f_1(x) + \frac{1}{2} f_2(x)$ $(a = 1)$

1. $f_1(x) = e^{-x}$ définie sur $[0, +\infty[$
2. $f_2(x) = e^{x}$ définie sur $]-\infty, 0]$



1. générer u_1 et u_2 uniformes entre 0 et 1
2. si $u_1 < 0.5$, on retourne $-ln(u2)$, sinon on retourne $ln(u2)$

Convolution

Hypothèse : la variable aléatoire X à générer peut s'écrire sous la forme de somme de n variables aléatoires plus faciles à générer :

$$X = X_1 + X_2 + \ldots + X_n$$

La méthode consiste alors à simplement générer les n valeurs x_i et retourner la somme

Exemples:

- Une Binomiale $B(n, p)$ est la somme de n Bernoulli de paramètre p
- Une Erlang k est la somme de k Exponentielles de même paramètre
- Une χ^2 à n degrés de liberté est la somme des carrées de n Normales centrées réduites

Caractérisation statistique

Les caractéristiques spécifiques de certaines distributions permettent de les générer par des algorithmes particuliers

Exemple : Méthode de Box et Muller pour générer une distribution Normale $N(\mu, \sigma)$:

1. générer U_1 et U_2 uniformes et indépendantes
2. $X_1 = \mu + \sigma\ cos(2\pi\ U_1)\sqrt{-2ln(U_2)}$ et
   $X_2 = \mu + \sigma\ sin(2\pi\ U_1)\sqrt{-2ln(U_2)}$ suivent une loi Normale $N(\mu, \sigma)$ et sont indépendantes

Quelques distributions souvent utilisées

- Normale
- Exponentielle
- Weibull
- Pareto
- Bernoulli
- Binomiale
- Géométrique
- Poisson

Normale

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \text{avec} \quad -\infty \leq \text{x} \leq +\infty.$$

μ est la moyenne et σ l'écart type ($\sigma \geq 0$)

La fonction de répartition n'est pas explicite, il faut recourir à des tables ou à des approximations numériques

La loi Normale est probablement la loi la plus utilisée à cause du théorème central Limite :

Théorème La somme de variables aléatoires indépendantes et de variance finie est assymptotiquement distribuée selon une loi Normale

Remarque Il n'est pas nécessaire que les variables aient la même distribution

Utilisation: modéliser un phénomène aléatoire résultant de la somme de plusieurs effets indépendants, par exemple :

- nombre de pannes sur un parc de matériel
- erreur de mesure ou de modélisation provenant de plusieurs facteurs, l'erreur autour de la vraie valeur sera distrbuée selon la loi normale

La loi Normale est une loi limite pour les opérations Somme et pour de nombreuses distributions quand un de leurs paramètres augmente (Binomiale, Poisson, ...)

simulation : méthode de Box et Muller ou somme d'un grand nombre de valeurs uniformes indépendantes

Exponentielle

$$f(x) = \lambda e^{-\lambda x}, \quad \text{avec} \;\; 0 \leq \mathrm{x} \leq \infty.$$

Seule loi continue vérfiant la propriété “Sans-Mémoire” :

$$P(X \leq t_0 + t | X > t_0) = P(X \leq t)$$

La propriété “sans mémoire” simplifie beaucoup l’analyse de modèles, ce qui explique la grande popularité de cette loi

Utilisée en général pour modéliser le temps entre deux événements succesives, par exemple début et fin d’un service, inter-arrivées de paquets, de requêtes, de pannes ...

simulation : méthode de transformation inverse

Weibull

$$f(x) = \frac{bx^{b-1}}{a^b} e^{-\left(\frac{x}{a}\right)^b} \quad \text{avec} \quad 0 \leq \text{x} \leq \infty.$$

pour $b = 1$, on retrouve la loi exponentielle

Loi limite sur les opérations Min

le minimum de plusieurs variables aléatoires i.i.d converge assymptotiquement vers une loi de Weibull

Utilisée souvent en fiabilité pour modéliser des durées de vie

Durée de Vie (Système) = Min (Durée de Vie Composante)

simulation : méthode de transformation inverse

Pareto

$$f(x) = \frac{ab^a}{x^{a+1}} \quad \text{avec} \quad \text{x} \geq \text{b}, \quad \text{a, b} > 0$$

Il a été trouvé que certaines caratéristiques du trafic Internet peuvent être bien représentées par cette loi, par exemple :

- taille des fichiers échangés
- durées des sessions FTP

Permet de générer facilement un trafic self-similar (qui permet de modéliser le trafic TCP/IP). En effet, il suffit d'aggréger plusieurs sources ON-OFF avec des distributions de Pareto pour la longeur des périodes ON-OFF

simulation : méthode de transformation inverse

Bernoulli

Distribution discrète la plus simple.

Une variable X de type Bernoulli $B(p)$ prend deux valeurs seulement 0 et 1:

$$Prob(X = 0) = p, \quad Prob(X = 1) = 1 - p$$

utilisée pour modéliser deux alternatives, par exemple : système en marche ou en panne, transmission avec succés ou non, ...

simulation : générer u uniforme entre 0 et 1, si $u \leq p$ retourner 0, sinon retourner 1

Binomiale

$$Prob(X = k) = C_n^k p^k (1-p)^{n-k}, \quad k = 0, 1, \ldots, n$$

utilisée pour modéliser le nombre de succés dans une suite de n tirages de Bernoulli indépendents, par exemple :

- nombre de processeurs en marche dans une machine multi-processeurs
- nombre de paquets qui ont atteint leur destination

simulation : méthode de convolution (somme de Bernoulli)

Géométrique

$$Prob(X = k) = p(1-p)^{k-1}, \quad k = 0, 1, \ldots, \infty$$

équivalent discret de la loi exponentielle, propriété "sans mémoire" :

$$P(X = k + k_0 | X > k_0) = P(X = k)$$

utilisée pour modéliser un nombre d'essais avant le premier succés ou le premier échec, par exemple :

nombre de paquets transmis avec succès entre deux paquets transmis avec erreur

simulation : méthode de transformation inverse

Loi de Poisson

$$Prob(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}, \quad k = 0, 1, \ldots, \infty$$

utilisée pour modéliser un nombre d'arrivées durant un intervalle de temps, par exemple :

nombre de requêtes à une base de donnée durant un intervalle de longeur t

simulation : méthode de transformation inverse (voir cours précédent)

Processus de Poisson

- Un processus de Poisson (flux Poissonien) est un processus tel que les différences entre deux instants successifs sont i.i.d de loi exponentielle ($Exp(\lambda)$)
- Lien avec Loi de Poisson : nombre d'arrivées durant un intervalle de temps de longeur t suit une loi de Poisson de paramètre λt
- La superposition de plusieurs flux Poissoniens de paramètres λ_i est un flux Poissonien de paramètre $\lambda = \sum \lambda_i$
- Si un flux Poissonien de paramètre λ est dispatché en k flux avec les probabilités $(p_1, p_2, \ldots, p_k)$ alors chaque flux i est aussi un flux Poissonien de paramètre $p_i\lambda$

Analyse opérationnelle

Quelques lois simples sur des quantités mesurables, qui ne nécessitent pas des hypothèses probabilistes ou statistiques :

- Loi de l'utilisation
- Loi du flux forcé
- Loi de Little
- Loi général du temps de réponse
- Loi du temps de réponse interactive

Quantités opérationnelles concernées

Pour une période d'observation T et un composant i du système, on peut mesurer les quantités :

- Taux d'arrivées λ_i: nombre d'arrivées sur le temps total T : $\frac{A_i}{T}$
- Débit X_i: nombre de travaux terminés sur le temps total T : $\frac{C_i}{T}$
- Utilisation U_i: temps total des services sur le temps total T : $\frac{B_i}{T}$
- Temps moyen de service S_i: temps total des services sur le nombre de travaux terminés: $\frac{B_i}{C_i}$

Loi de l'utilisation

- Par définition

$$U_i = \frac{B_i}{T} = \frac{C_i}{T} \times \frac{B_i}{C_i}$$

$$\boxed{U_i = X_i S_i}$$

- L'utilisation est le produit du débit et du temps moyen de service

Loi du flux forcé

- Relie le débit X du système aux débits X_i des éléments individuels
- Hypothèse : conservation des flux entrées sorties: $A_i = C_i$ (par exemple, quand T devient grand et le système est stable)
- Dans le système, un travail qui arrive se décompose en tâches élémentaires et ne le quitte que lorsqu'elles sont toutes terminées
- Il demande V_i tâches au serveur i (V_i visites au serveur i)
- C_0 est le nombre de travaux entrant et quittant le système
- Donc le serveur i reçoit et effectue $C_i = V_i C_0$ tâches
- Le débit du serveur $X_i = \frac{C_i}{T} = \frac{C_0}{T}\frac{C_i}{C_0}$, donc

$$\boxed{X_i = XV_i}$$

- Puisque l'utilisation U_i vaut par définition $X_i S_i$, on a:

$$U_i = X_i S_i = X V_i S_i$$

- En posant $D_i = V_i S_i$, on obtient

$$\boxed{U_i = X D_i}$$

- D_i est le temps de service dans le serveur i de toutes les tâches d'un travail
- Le goulot d'étranglement (bottleneck) est le serveur qui a la plus grande valeur de D_i et donc l'utilisation la plus élevée
- le nombre maximal de travaux exécutables dans le système est obtenu lorsque le serveur en bottleneck a une utilisation de 1

Exemple

On considère un système composé d'un processeur et deux disques A et B. on suppose que :

- chaque programme nécessite 5 secondes de temps CPU, 80 accès sur le disque A et 100 accès sur le disque B
- un accès disque sur A dure 50 ms, et sur B 30 ms
- on mesure un débit de 15.7 accès par seconde sur le disque A
- on veut déterminer le bottleneck et calculer le débit et l'utilisation de la CPU et des disques?

Données initiales

- $D_{CPU} = 5s$
- $V_A = 80$
- $V_B = 100$
- $S_A = 0.05s$
- $S_B = 0.03s$
- $X_A = 15.7$

- $D_A = S_A V_A = 4s$
- $D_B = S_B V_B = 3s$
- donc la CPU est le bottleneck
- $X = \frac{X_A}{VA} = 0.1963$ programme par seconde
- $X_B = XV_B = 19.63$ accès par seconde
- $X_{CPU} = XV_{CPU} = 0.1963 \times 181 = 35.48$ vistes par seconde
  (puisqu'un programme doit visiter la CPU avant d'accéder aux disques, le nombre de visites à la CPU est $V_{CPU} = V_A + V_B + 1 = 181$)
- $U_{CPU} = XD_{CPU} = 0.1963 \times 5 = 98\%$
- $U_A = XD_A = 0.1963 \times 4 = 78.4\%$
- $U_B = XD_B = 0.1963 \times 3 = 58.8\%$

Loi de Little

Hypothèse : nombre de clients entrants = nombre de clients sortants

- $\overline{N}$ est le nombre moyen de clients dans le système
- $\overline{R}$ est le temps moyen de réponse (temps d'attente + temps de service)
- λ est le taux d'arrivée des clients dans le système

$$\overline{N} = \lambda \overline{R}$$

ou

$$\overline{N} = X \overline{R}$$

Remarque : La loi de Little peut s'appliquer à n'importe quelle partie d'un système

- Calcul du nombre moyen de clients en attente (file d'attente sans le serveur) :

$$\overline{N_w} = \lambda \overline{W}$$

où $\overline{W}$ est le temps moyen d'attente

- Calcul du nombre moyen de clients en service (serveur (s) sans la file) :

$$\overline{N_s} = \lambda \overline{S}$$

où $\overline{S}$ est le temps moyen de service

Loi général du temps de réponse

On considère un système composé de M sous-systèmes

- Par la loi de Little on a $Q_i = X_i R_i$ et $Q = XR$ où :
  - Q_i (resp Q) est le nombre moyen de clients dans le sous-système i (resp dans le système)
  - R_i (resp R) temps moyen de réponse du sous-système i (resp du système)
- $Q = Q_1 + \ldots + Q_M$, donc $XR = X_1 R_1 + \ldots + X_M R_M$ et

$$R = \sum_{i=1}^{M} V_i R_i$$

Remarque: il est possible de montrer que cette loi est valide même sans l'hypothèse de conservation de flux

Exercices

Exercice 1 Le débit d'un système a été observé être égale à 5 clients par seconde sur une période d'observation de 10 minutes. Si le nombre moyen de clients dans le système a été égal à 4 durant cette période, quel était le temps moyen de réponse?

Exercice 2 Durant une période d'observation de 10 secondes, 40 requêtes ont été traitées par un serveur. Chaque requête a nécessité deux accès disque. Le temps moyen de service sur le disque était de 30ms. Quel a été le taux d'utilisation du disque durant cette période?

Chaînes de Markov

Une chaîne est un processus à valeurs dans un espace d'états E dénombrable

- On distingue :
  - les Chaînes de Markov en Temps Discret (CMTD) :
  - les Chaînes de Markov en Temps Continu (CMTC) :
- Définition : Un processus X_n est une CMTD ssi

  $Pr(X_{n+1} = j | X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = Pr(X_{n+1} = j | X_n = i_n)$

CMTD homogène

On note $P_{i,j}(n) = Pr(X_{n+1} = j | X_n = i)$

- Une chaîne de Markov est homogène, si est seulement si, $P_{i,j}(n)$ est indépendant de la date n. Cette probalilité, noté $P_{i,j}$, est la probabilité de transition de i vers j

- Par construction, la matrice $P = (P_{i,j})_{i,j \in E}$ des probabilités de transition vérifie :
  - tous les éléments de P sont positifs ou nuls
  - pour tout i, $\sum_{j \in E} P_{i,j} = 1$

Comportement transitoire et limite

- Soit $\pi(n)$ le vecteur des probabilités $Pr(X_n = i)$
- En appliquant le théorème de conditionnement
  $Pr(X_n = j) = \sum_{i \in E} Pr(X_n = j | X_{n-1} = i) Pr(X_{n-1} = i) =$
  $\sum_{i \in E} P_{i,j} Pr(X_{n-1} = i)$
- Ainsi, $\pi(n) = \pi(n-1)P$ et par récurrence $\pi(n) = \pi(0)P^n$
- Si le processus est connu à l'instant 0, on peut calculer ainsi les distributions transitoires aux instants n
- Pour étudier l'existence d'une limite au vecteur $\pi(n)$ lorsque le temps n tend vers l'infini, il faut établir une classification des états et analyser la structure de la chaîne

Classification des Etats

On définit pour chaque état quatre quantités liées au dates de retour à cet état.

- f_j^n est la probabilité de l'événement "(le premier retour en j à lieu en n transitions)".
- f_j est la probabilité de retour en j : $f_j = \sum_{n=1}^{\infty} f_j^n$
- M_j est le temps moyen de retour en j : $M_j = \sum_{n=1}^{\infty} n \times f_j^n$
- γ est le pgcd des valeurs de n telles que f_j^n est non nulle. γ est la période de retour.

Transience ou Récurence

- Si $f_j < 1$ l'état est transitoire.
- Si $f_j = 1$ l'état est récurent :
  - si $M_j = \infty$, l'état est recurent nul.
  - si $M_j < \infty$, l'état est recurent non nul.

  De plus,
  - si $\gamma > 1$, l'état est périodique de période γ.
  - si $\gamma = 1$, l'état est apériodique.

Structure de la chaîne

- Définition : Un sous ensemble A d'états est fermé si et seulement si il n'y a pas de transition entre cet ensemble et son complémentaire
- Définition : un état absorbant est un sous ensemble fermé ne comprenant qu'un seul élément.
- Une chaîne est irréductible si et seulement si il existe une suite de transitions menant de i à j pour tous les états i et j.

Chaînes irréductibles

- **Théorème:** Pour une chaîne de Markov irréductible, tous les états sont:
  - tous transitoires
  - ou tous récurents non nuls
  - ou tous récurents nuls

  De plus, si un état est périodique, alors tous les états le sont avec la même période.
- **Remarque:** Pour une chaîne finie irréductible, tous les états sont récurents non nuls

Existence d'une limite

Soit une CMTD irréductible et apériodique, alors la distribution limite $\pi = lim_{n\to\infty}\pi(n)$ existe et est indépendente de la distribution initiale $\pi(0)$ (on dit qu'elle est ergodique). De plus,

- soit tous les états sont transitoires ou récurents nuls et dans ce cas $\pi(j) = 0$, pour tout j.
- soit tous les états sont récurents non nuls et π est solution unique du système :

$$\begin{cases} \pi = \pi P \\ \pi e = 1 \end{cases} \tag{5}$$

où e est un vecteur colonne dont tous les éléments sont 1. De plus, on a $\pi(j) = 1/M_j$

Chaînes de Markov en Temps Continu (CMTC)

diffèrent des CMTD par les transitions entre états qui peuvent occurer à des instants arbitraires dans le temps

- Définition : Un processus $X(t)$ est une CMTC ssi
  $Pr(X(t_n) = e_n | X(t_{n-1}) = e_{n-1}, X(t_{n-2}) = e_{n-2}, \ldots, X(t_0) = e_0) =$
  $Pr(X(t_n) = e_n | X(t_{n-1}) = e_{n-1}), \quad \forall n, \ \forall t_0 < t_1 < \ldots < t_n$
- En pratique : il faut s'assurer que la description du système à un instant t est suffisante pour prédire le futur

CMTC homogènes

- On s'intéresse uniquement aux CMTC homogènes :
  $Pr(X(t_n) = j | X(t_{n-1}) = i)$ ne dépend pas des instants t_{n-1} et t_n mais seulement de la durée $t_n - t_{n-1}$
- On définit alors la probabilité de transition de i à j durant le temps t
  $p_{i,j}(t) = Pr(X(t+s) = j | X(s) = i) = Pr(X(t) = j | X(0) = i),\ \forall s \geq 0$
- Si $\pi(t)$ est la distribution transitoire à l'instant t, càd, le vecteur des probabilités $Pr(X(t) = i)$, alors $\pi_j(t) = \sum_i p_{i,j}(t)\pi_i(0)$
- Soit $P(t)$ la matrice des transitions $p_{i,j}(t)$, on a $\pi(t) = \pi(0)P(t)$

Remarque : Pour une CMTC homogène le temps passé dans un état a une distribution exponentielle

Générateur d'une CTMC

- On définit $q_{i,j}$ le taux de transition instantané de i à j par :

$$q_{i,j} = lim_{dt \to 0} \frac{p_{i,j}(dt)}{dt}, \quad (i \neq j)$$

- On pose $q_{i,i} = -\sum_{\{j,\ j \neq i\}} q_{i,j}$
- La matrice $Q = (q_{i,j})$ est appelée générateur infinitésimal de la CMTC $X(t)$, elle vérifie les propriétés :
  - tous les éléments non diagonaux de Q sont positifs ou nuls
  - la somme de chaque ligne est nulle ($\forall i, \quad \sum_j q_{i,j} = 0$)
- On peut montrer que $P(t) = e^{tQ}$ et donc $\pi(t) = \pi(0)e^{tQ}$

Caractéristiques d'une CMTC

- La classification des états d'une CMTC en états transitoires, récurrents nuls ou récurrents non nuls se fait de la même façon que pour les CMTD
- La notion de périodicité n'existe pas pour les CMTC
- Une CMTC est dite irréductible si chaque état j est atteignable de n'importe quel état i $\forall i, j, i \neq j, \exists t : p_{i,j}(t) > 0$
- Si une CMTC est irréductible, alors tous les états sont de même nature

Remarque : la méthode de la *chaîne incluse* permet d'étudier les caractéristiques d'une CMTC à partir de celles de la CMTD incluse

Existence d'une limite

Théorème: pour une CMTC irréductible la distribution limite $\pi = lim_{t\to\infty}\pi(t)$ existe et est indépendante de la distribution initiale $\pi(0)$. De plus,

- soit tous les états sont transitoires ou récurrents nuls et dans ce cas $\pi_j = 0$, pour tout j.
- soit tous les états sont récurrents non nuls et π est l'unique solution du système (dans ce cas on dit que la chaîne est ergodique) :

$$\begin{cases} \pi Q = 0 \\ \pi e = 1 \end{cases}$$

  où e est un vecteur colonne dont tous les éléments sont égaux à 1
- La solution π de ce système est appelée distribution stationnaire

Equilibre stationnaire

- En développant l'équation $\pi Q = 0$ pour un i quelconque, on a:

$$\sum_j \pi_j q_{j,i} = 0$$

- En séparant la somme et en utilisant le fait que $\sum_k q_{i,k} = 0$:

$$\sum_{j \neq i} \pi_j q_{j,i} = -\pi_i q_{i,i} = \pi_i \sum_{k \neq i} q_{i,k}$$

- On obtient ce qu'on appelle l'équation d'équilibre ou les équations de balance globale :

$$\boxed{\sum_{j \neq i} \pi_j q_{j,i} = \sum_{k \neq i} \pi_i q_{i,k}}$$

  Interprétation : tout ce qui permet de rentrer dans l'état i (partie gauche) est égal à tout ce qui permet de sortir de i (partie droite)

Lien entre CMTC et CMTD : Uniformisation

On peut transformer les matrices pour passer d'un problème en temps continu en un problème en temps discret

- Soit Q un générateur tel que $max_i(|q_{i,i}|) < +\infty$
- Définissons $P_\lambda = Id + \frac{1}{\lambda}Q$
- Pour $\lambda \geq max_i(|q_{i,i}|)$ on peut montrer que :
  - P_λ est une matrice stochastique qui a la même distribution stationnaire que Q
  -

$$P(t) = e^{-\lambda t} \sum_{n=0}^{\infty} \frac{(\lambda t)^n}{n!} {P_\lambda}^n, \quad t \geq 0$$

Calcul de la distribution stationnaire

- Résolution analytique : pour certains modèles (voir plus loin)
- Résolution numérique : limitée par le problème de l'explosion de la taille de l'espace d'états. Les principales méthodes de résolution sont:
  - Méthodes directes (variantes de l'élimination de Gauss)
  - Méthodes itératives (puissances, Jacobi, Gauss-Seidel, ...)
  - Méthodes par décomposition (NCD, complément stochastique, ...)
- En cas de besoin, le livre de Stewart "Introduction to the Numerical Solution of Markov Chains" est très complet sur ce sujet

Un exemple de résolution analytique : Processus de Naissance et de Mort

- Un processus de Naissance et de Mort est une chaîne de Markov où pour tout état n, les seules transitions possibles amènent aux états $n-1$ et $n+1$ (s'ils existent)
- La transition de l'état n à l'état $n+1$ est une Naissance
- La transition de l'état n à l'état $n-1$ est une Mort
- λ_n (resp. μ_n) est le taux de Naissance (resp. de Mort) à l'état n

Equilibre stationnaire

pour $n > 0$:

$$\pi_n[\mu_n + \lambda_n] \;=\; \pi_{n-1}\lambda_{n-1} + \pi_{n+1}\mu_{n+1}$$

pour $n = 0$

$$\pi_0\lambda_0 \;=\; \pi_1\mu_1$$

oubien

$$\pi_n[\mu_n 1_{\{n>0\}} + \lambda_n] \;=\; \pi_{n-1}\lambda_{n-1}1_{\{n>0\}} + \pi_{n+1}\mu_{n+1}$$

Résolution

- L'équation à l'état 0 permet d'obtenir π_1 en fonction de π_0

$$\pi_0\lambda_0 \;=\; \pi_1\mu_1$$

- Examinons maintenant l'équation à l'état 1

$$\pi_1[\mu_1 + \lambda_1] \;=\; \pi_0\lambda_0 + \pi_2\mu_2$$

- Après simplification : $\pi_1\lambda_1 \;=\; \pi_2\mu_2$
- Par récurrence sur n :

$$\pi_n\lambda_n \;=\; \pi_{n+1}\mu_{n+1}$$

- Soit:

$$\pi_n \;=\; \pi_0 \frac{\prod_{i=0}^{n-1} \lambda_i}{\prod_{i=1}^{n} \mu_i}$$

- Si on peut normaliser:

$$\sum_{n=1}^{\infty} \frac{\prod_{i=0}^{n-1} \lambda_i}{\prod_{i=1}^{n} \mu_i} < \infty$$

- alors,

$$\pi_0 = \left[1 + \sum_{n=1}^{\infty} \frac{\prod_{i=0}^{n-1} \lambda_i}{\prod_{i=1}^{n} \mu_i} \right]^{-1}$$

Une application des chaînes de Markov : Google

- Comment classer les pages Web pour trouver les plus pertinentes?
- L'outil qui permet de faire cela est l'algorithme **PageRank** proposé par Sergy Brin and Larry Page dans leur article "PageRank: Bringing order to the Web" en 1998
- En exploitant cet algorithme, Sergy Brin and Larry Page ont fondé par la suite l'entreprise qui est derrière le célèbre moteur de recherche **Google**

Fondamentaux de PageRank

- PageRank calcule une valeur numérique pour chaque page Web. Cette valeur mesure la pertinence ou l'importance relative d'une page dans un ensemble de pages Web
- une page Web est un texte caractérisé par des liens vers d'autres pages
- ainsi chaque page est pointée par un ensemble de pages et pointe elle même vers d'autres pages. Soit $\Gamma(p)$ l'ensemble des pages qui pointent sur p et $d(q)$ le nombre de pages pointées par la page q
- si une page source pointe vers k pages destinations, alors chacune reçoit $1/k$ de la pertinence de la source
- si $Pe(p)$ est la pertinence de la page p, alors :
  $Pe(p) = \sum_{q \in \Gamma(p)} \frac{Pe(q)}{d(q)}$

Modélisation par une chaîne de Markov

- l'espace d'états est constitué par l'ensemble des pages Web $\{p_1, p_2, \ldots, p_N\}$
- les transitions entre états sont les liens entre les pages Web
- on peut alors définir la matrice de transitions $H = H_{i,j}$:
  - $H_{i,j} = 1/O_i$ si p_i a un lien vers p_j et 0 sinon
  - O_i est le nombre de pages pointées par p_i
- La chaîne de Markov (X_n) peut être vue comme une personne qui surfe sur le Web de façon aléatoire et qui peut se trouver à l'étape n à une des pages Web
- Les pages qui n'ont pas de liens vers d'autres pages sont supposées pointer vers l'ensemble des pages Web. Ceci permet d'éviter les états absorbants qui pourraient arrêter le processus du surf

- En utilisant ce qui précède, on a :

$$Pe(p_j) = \sum_{p_i \in \Gamma(p_j)} \frac{Pe(p_i)}{O_i} = \sum_{i=1}^{N} Pe(p_i) H_{i,j}$$

- On reconnaît la définition de la probabilité stationnaire d'une chaîne de Markov
- La pertinence d'une page p_i est donc égale à la probabilité stationnaire π_i de cette page, qui peut être interprétée comme la probabilité de se trouver dans cette page après plusieurs clics
- Rappel : $\pi_i = 1/\mu_i$ où μ_i est le temps moyen de retour à la page p_i
- une page avec une grande valeur π_i doit être importante, car cela implique que plusieurs pages pointent vers elle et font ainsi que la personne qui surfe y retourne assez souvent

Remarques :

- En fait, la matrice de transition G utilisée dans PageRank (appelée Matrice de Google) est légèrement différente : $G = \alpha H + (1 - \alpha)U$, où :
  - H est la matrice de transition déjà définie
  - U est une matrice dont toutes les valeurs sont égales à $1/N$
  - α est un paramètre entre 0 et 1, souvent choisi égale à 0.85
- L'idée de cette modification est que dans 85% des cas, on surfe selon la structure des liens du Web et dans 15% des cas on choisit une nouvelle page de façon uniforme dans l'ensemble des pages
- Pour plus de détails sur PageRank et Google l'information est largement disponible sur le Web (par exemple sur Wikipedia)

Files d'attentes :Notation de Kendall

On décrit une file de façon compacte avec la notation **A/S/n/B/N/Z**:

A décrit le processus des arrivées (ou distribution d'inter-Arrivées). Les symboles suivants sont souvent utilisés : (M : processus de Poisson), (GI : lois générales indépendantes), (G: Général), (D: Déterministe), (E_k: Erlang d'ordre k)

S décrit les distributions de service. On utilise les mêmes symboles que pour les arrivées (M signifie services exponentiels)

n est le nombre de serveurs

B est la capacité du Buffer, par défaut la file est de taille infinie

N est le nombre de clients existant dans le système (source + file), par défaut, la population est de taille infinie

Z discipline de service, par défaut c'est FIF0

La file M/M/1

- Arrivées Poisson, Services Exponentiels, 1 Serveur, Capacité infinie pour garder les clients
- On note μ le taux de service et λ le taux d'arrivées
- C'est un processus de Naissance et de Mort avec les taux de transition :

$$\begin{cases} \lambda_i = \lambda & \forall i \\ \mu_i = \mu & \forall i \end{cases}$$

- Posons $\rho = \lambda/\mu$ (c'est la charge du système)

CMTC associée à une file M/M/1

- Soit N_t le nombre de clients dans la file à l'instant t
- Le processus $(N_t)_{t\geq 0}$ à espace discret $\{0, 1, 2, \ldots\}$ et à temps continu est une CMTC. Ceci est dû au fait que les inter-arrivées et les services sont exponentielles (sans mémoire)
- Si on considère par exemple la file M/D/1, $(N_t)_{t\geq 0}$ n'est pas une chaine de Markov: il faut connaitre de plus le temps passé déjà dans le service pour prédire un départ
- Générateur : $Q = \begin{pmatrix} -\lambda & \lambda & 0 & \ldots & \ldots \\ \mu & -\nu & \lambda & 0 & \ldots \\ 0 & \mu & -\nu & \lambda & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$ $(\nu = \lambda + \mu)$

- Equation d'équilibre stationnaire :

$$\pi_n[\mu 1_{\{n>0\}} + \lambda] \;=\; \pi_{n-1}\lambda 1_{\{n>0\}} + \pi_{n+1}\mu$$

- Ce qui donne :

$$\pi_n \;=\; \pi_0\rho^n, \quad \forall n \geq 0$$

- Condition de normalisation : $\pi e = 1 \Longleftrightarrow \pi_0 \sum_{n=0}^{\infty} \rho^n = 1$. Ceci est possible, ssi $\sum_{n=0}^{\infty} \rho^n < +\infty \Longleftrightarrow \rho < 1$
- $\rho < 1 (\Longleftrightarrow \lambda < \mu)$ est appelée condition de stabilité du système. Intuitivement, pour que le système soit stable, il ne faut pas qu'il arrive plus de clients que ce qu'il est capable de traiter
- Sous la condition $\rho < 1$, les probabilités stationnaires sont données par : $\pi_n = (1-\rho)\rho^n, \quad n \geq 0$
- Remarque : on peut montrer que la chaine est ergodique, ssi $\rho < 1$

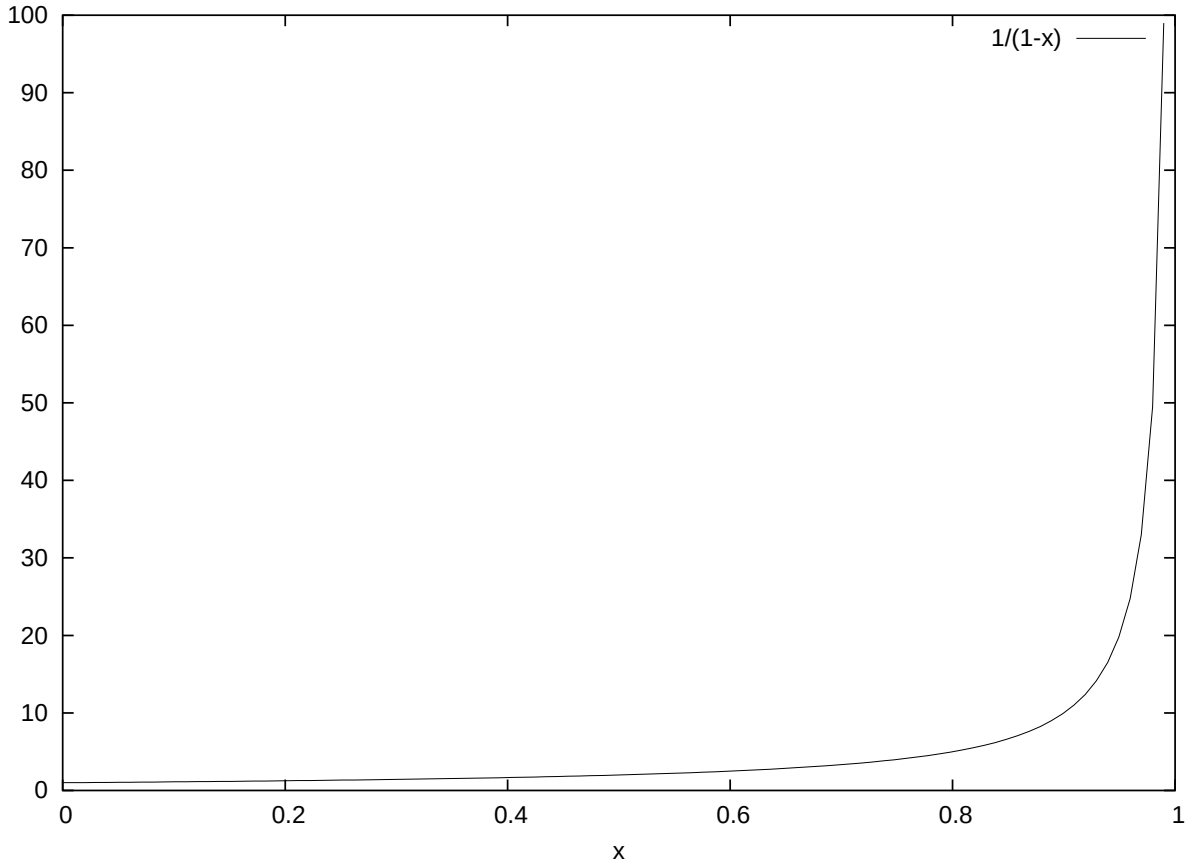
Caractéristiques d'une M/M/1

- La moyenne du nombre de clients $E(N)$ dans la file:
  $E(N) = \sum_{n=1}^{\infty} n\pi_n = \frac{\rho}{1-\rho}$
- La probabilité que la file soit vide est $(1 - \rho)$
- le taux d'utilisation de la file U vaut donc ρ
- Variance $V(N)$ du nombre de clients dans la file:
  $V(N) = E(N^2) - (E(N))^2 = \sum_{i=1}^{\infty} i^2\pi_i - (E(N))^2 = \frac{\rho}{(1-\rho)^2}$
- Probabilité qu'il y ait au moins n clients dans la file: ρ^n
- La moyenne du nombre de clients en attente $E(N_W)$:
  $E(N_W) = \sum_{n=1}^{\infty} n\pi_{n+1} = \frac{\rho^2}{1-\rho}$

Caractéristiques temporelles

- En appliquant la formule de Little :
  - le temps moyen de réponse est : $E(R) = \frac{E(N)}{\lambda} = \frac{1}{\mu - \lambda} = \frac{1}{\mu(1-\rho)}$
  - le temps moyen d'attente est : $E(W) = \frac{E(N_W)}{\lambda} = \frac{\rho}{\mu - \lambda} = \frac{\rho}{\mu(1-\rho)}$

    On peut remarquer que ces temps tendent vers l'infini quand λ tend vers μ
- On peut montrer que :
  - le temps de réponse a une distribution exponentielle, la fonction de répartition est : $F_R(x) = Prob(R \leq x) = 1 - e^{-x(\mu-\lambda)}$
  - le temps d'attente est distribué selon la fonction de répartition : $F_W(x) = Prob(W \leq x) = 1 - \rho e^{-x(\mu-\lambda)}$



Allure du temps moyen d'attente et de séjour en fonction de ρ

Non Linéarité

Le temps moyen de réponse (et d'attente) est fortement non linéaire :

- On considère une file dont la durée de service moyenne est $144s$
- Le taux d'arrivée est 10 clients/h
- Que vaut le temps moyen de réponse ?
- le temps moyen de service est $0.04h$
- Donc le taux de service est 25 clients/h et la charge vaut 0.4
- Et le temps de réponse vaut en seconde $144/(1 - 0.4) = 240s$
- Le taux d'arrivées augmente de 2 clients/h
- La charge passe à 0.48 et le délai passe à $277s$
- Augmenter les arrivées de 20% augmente le délai de 17%

- Le taux d'arrivée est maintenant de 20 clients/h
- Donc la charge vaut 0.8
- Et le temps de réponse vaut en seconde $720s$
- Le taux d'arrivées augmente de 2 clients/h
- La charge passe à 0.88
- Le délai passe à $1200s$
- Augmenter les arrivées de 10% augmente le délai de 67%

Compromis Usager/Opérateur

- Un opérateur souhaite que son système soit utilisé le plus possible
- donc ρ doit être grand
- Mais lorsque ρ s'approche de 1, le temps d'attente des usagers est non borné
- Demandes antagonistes : compromis....

Dimensionnement

- Déterminer la capacité du service pour garantir un délai voulu pour un trafic (taux d'arrivée) donné
- La capacité C est le débit du serveur $= \mu$
- Soit R le temps de réponse souhaité

$$R = \frac{1}{C - \lambda}$$

- Donc, $C = \lambda + 1/R$
- $C \geq \lambda$ assure la stabilité du système, le terme $1/R$ est le coût pour obtenir les performances demandées sur un système simple donné

La file M/M/1	
Caractéristiques Spatiales	
Charge	$\rho = \lambda/\mu$
Probabilité d'avoir k clients dans la file	$\pi_k = (1-\rho)\rho^k$
Utilisation	$U = \rho$
Nombre moyen de clients dans la file	$E(N) = \frac{\rho}{(1-\rho)}$
Variance du nombre de clients dans la file	$V(N) = \frac{\rho}{(1-\rho)^2}$
Probabilité d'avoir au moins n clients	ρ^n
Nombre moyen de clients en attente	$E(N_W) = \frac{\rho^2}{(1-\rho)}$
Variance du nombre de clients en attente	$V(N_W) = \frac{\rho^2(1+\rho-\rho^2)}{(1-\rho)^2}$

Caractéristiques Temporelles	
Répartition du temps de réponse	$F(x) = 1 - e^{-x(\mu-\lambda)}$
Temps de réponse Moyen	$E(R) = \frac{1}{(\mu-\lambda)}$
Variance du temps de réponse	$V(R) = \frac{1}{(\mu-\lambda)^2}$
Répartition du temps d'attente	$F(x) = 1 - \rho e^{-x(\mu-\lambda)}$
Temps d'attente moyen	$E(W) = \frac{\rho}{\mu-\lambda}$
Variance du temps d'attente	$V(W) = \frac{\rho(2-\rho)}{(\mu-\lambda)^2}$

Processus de Sortie d'une M/M/1

- On peut caractériser le processus de sortie d'une file M/M/1 :
- **Théorème de Burke** Le processus de sortie d'une M/M/1 de taux d'arrivée λ et de taux de service μ est un processus de Poisson de taux λ si $\lambda < \mu$
- Utile dans les méthodes de décomposition de réseaux ouverts

Exemple M/M/1 : Requêtes dans un SGBD

- Arrivées selon un processus de Poisson de taux 30 req par seconde.
- Chaque requête prend 0.02 seconde de traitement
- Donc le taux de service est de $1/0.02 = 50$ requetes par seconde
- Analyse par une M/M/1
- La charge est de 0.6
- Le serveur est vide pendant 40 pourcent du temps
- Le nombre moyen de requêtes est 1.5
- Donc le temps moyen de réponse est 0.05 seconde

Files simples, Modèles d'Erlang

Modèle d'Erlang C

- m canaux
- Arrivées Poisson, Services Exponentiels
- Capacité de stockage inifini
- On veut calculer la probabilité γ que les m canaux (serveurs) soient occupés
- C'est aussi la probabilité qu'un client entrant soit contraint d'attendre avant de commencer son service

Ce modèle correspond à une file M/M/m

La file M/M/m

- C'est un processus de Naissance et de Mort caractérisé par:

$$\begin{cases} \lambda_i = \lambda & \forall\ i \\ \mu_i = i\mu & si \quad i < m \\ \mu_i = m\mu & si \quad i \geq m \end{cases}$$

- La condition de stabilité est : $\lambda < m\mu$ (dans la suite, on suppose que cette condition est vérifiée)
- On définit la charge par $\rho = \lambda/(m\mu)$
- On cherche à calculer $\gamma = \sum_{i \geq m} \pi(i)$

Formule d'Erlang C

En développant, l'équation déquilibre et la condition de normalisation,

- la probabilité d'avoir $k < m$ clients : $\pi_0 \frac{(m\rho)^k}{k!}$
- la probabilité d'avoir $k \geq m$ clients : $\pi_0 \frac{m^m \rho^k}{m!}$
- la probabilité d'attente (Formule d'Erlang C):

$$\gamma = \frac{(m\rho)^m}{m!(1-\rho)}\pi_0$$

où la probabilité que le système soit vide est :

$$\pi_0 = \left(1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{i=1}^{m-1} \frac{(m\rho)^i}{i!}\right)^{-1}$$

La file M/M/m	
Caractéristiques Spatiales	
Charge	$\rho = \lambda/(m\mu)$
Probabilité système vide	$\pi_0 = \left(1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{i=1}^{m-1} \frac{(m\rho)^i}{i!}\right)^{-1}$
Probabilité d'avoir k clients	si $k < m \quad \pi_0 \frac{(m\rho)^k}{k!}$
	sinon $\quad \pi_0 \frac{m^m \rho^k}{m!}$
Probabilité d'attente	$\gamma = \frac{(m\rho)^m}{m!(1-\rho)} \pi_0$
Nombre moyen de clients	$E(N) = m\rho + \frac{\rho\gamma}{1-\rho}$
Variance du nombre de clients	$V(N) = m\rho + \rho\gamma(m + \frac{1+\rho-\rho\gamma}{(1-\rho)^2})$
Nombre moyen de clients en attente	$E(N_W) = \frac{\rho\gamma}{(1-\rho)}$
Variance du nombre de clients en attente	$V(N_W) = \frac{\rho\gamma(1+\rho-\rho\gamma)}{(1-\rho)^2}$

La file M/M/m	
Caractéristiques Temporelles	
Temps de réponse Moyen	$E(R) = \frac{1}{\mu}(1 + \frac{\gamma}{m(1-\rho)})$
Variance du temps de réponse	$V(R) = \frac{1}{\mu^2}(1 + \frac{\gamma(2-\gamma)}{m^2(1-\rho)^2})$
Répartition du temps d'attente	$F(x) = 1 - \gamma e^{-x\mu m(1-\rho)}$
Temps d'attente moyen	$E(W) = \frac{\gamma}{m\mu(1-\rho)}$
Variance du temps d'attente	$V(W) = \frac{\gamma(2-\gamma)}{m^2\mu^2(1-\rho)^2}$

Modèle d'Erlang pour les blocages d'appel

- m canaux
- Arrivées Poisson, Service exponentiels
- Si il n'y a aucun canal libre, l'appel est perdu (blocage d'appel)
- On veut calculer la probabilité de blocage d'appels
- Ce modèle correspond à une file M/M/m/m
- Remarque : le système est toujours stable puisque le nombre d'états est fini

- La file M/M/m/m est un processus de naissance et mort avec les taux :

$$\begin{cases} \lambda_i = \lambda \quad \forall \quad i < m \\ \lambda_m = 0 \\ \mu_i = i\mu \qquad \forall i \end{cases}$$

- Puisque l'espace d'états est fini, la chaine est ergodique et les probabilités sationnaires sont données par :

$$\pi_k = \left(\sum_{i=0}^{m} \frac{\lambda^i}{i!\mu^i} \right)^{-1} \frac{\lambda^k}{k!\mu^k}, \ 0 \leq k \leq m$$

- On déduit alors la probabilité que le système soit plein π_m

Attention

- La probabilité de perte est la probabilité qu'un client entrant trouve le système plein
- En général, la distribution ψ aux instants d'arrivées n'est pas nécessairement la distribution stationaire π
- $\pi_i = limT(i)/T$ ($T(i)$ est le temps passé en i, T un temps d'observation très grand)
- $\psi_i = limX(i)/X$ ($X(i)$ est le nombre de fois où on a trouvé la file à l'état i, X le nombre de visites)
- Cependant, ici, grâce aux arrivées suivant un processus de Poisson, les deux distributions coincident
- C'est la propriété PASTA: Poisson Arrivals See Time Averages

Formule d'Erlang B

- La probabilité de perte vaut donc:

$$PrbPerte = \pi_m = \left(\sum_{i=0}^{m} \frac{\lambda^i}{i!\mu^i} \right)^{-1} \frac{\lambda^m}{m!\mu^m}$$

- C'est la formule d'Erlang B ou formule d'Erlang pour les blocages d'appel
- On peut alors dimensionner le commutateur (trouver m pour garantir une probabilité de blocage acceptable)

La file M/M/m/B

- Il y a B places et m serveurs ($B \geq m$)
- Le processus du nombre de clients est encore un processus de naissance et de mort dont les taux sont :

$$\begin{cases} \lambda_i = \lambda & \forall\ i < B \\ \lambda_B = 0 & \\ \mu_i = i\mu & si \quad i \leq m \\ \mu_i = m\mu & si \quad m < i \leq B \end{cases}$$

- Puisque la chaîne a un nombre fini d'états, le système est toujours stable

- Les probabilités stationnaires sont obtenues simplement :

$$\pi_n = \begin{cases} \frac{\pi_0 \lambda^n}{n!\ \mu^n} & \forall\ n < m \\ \frac{\pi_0 \lambda^n}{m!\ m^{n-m}\ \mu^n} & \forall\ n \geq m \end{cases}$$

- π_0 est obtenu par normalisation :

$$\pi_0 = \left(1 + \frac{(m\rho)^m (1 - \rho^{B+1-m})}{m!(1-\rho)} + \sum_{i=1}^{m-1} \frac{(m\rho)^i}{i!} \right)^{-1}$$

- Toutes les arrivées qui se produisent pendant que le buffer est dans l'état B sont perdues
- Le taux d'arrivée réel est $\lambda(1 - \pi_B)$ et le taux de perte est $\lambda \pi_B$
- La formule de Little donne le temps moyens de séjour et d'attente

$$E(R) = \frac{E(N)}{\lambda(1-\pi_B)} \quad et \quad E(W) = \frac{E(N_W)}{\lambda(1-\pi_B)}$$

La file M/M/m/B	
Charge	$\rho = \lambda/(m\mu)$
Probabilité d'avoir k clients	si $k \leq m \quad \pi_0 \frac{(m\rho)^k}{k!}$
	sinon $\quad \pi_0 \frac{m^m \rho^k}{m!}$
Nombre moyen de clients	$E(N) = \sum_{i=1}^{B} i\pi_i$
Nombre moyen de clients en attente	$E(N_W) = \sum_{i=m+1}^{B} (i-m)\pi_i$
Taux d'arrivée réel	$\lambda_r = \lambda(1-\pi_B)$
Utilisation moyenne d'un serveur	$\lambda_r/(m\mu) = \rho(1-\pi_B)$
Taux de perte	$\lambda\pi_B$
Temps de réponse moyen	$E(R) = \frac{E(N)}{\lambda(1-\pi_B)}$
Temps d'attente moyen	$E(W) = \frac{E(N_W)}{\lambda(1-\pi_B)}$

La file M/M/∞

- Arrivées Poisson de taux λ
- Un infinité de serveurs identiques (service exponentiel de taux μ)
- Pas de buffer d'attente, un client qui arrive entre directement en service
- Sert à modéliser des situations où le nombre de serveurs est plus grand que la demande
- Remarque : la capacité de service étant illimitée le système est toujours stable indépendemment des valeurs de λ et μ

Caractéristiques

- La file M/M/∞ est un processus de naissance et de mort avec les taux :

$$\begin{cases} \lambda_i = \lambda & \forall\ i \\ \mu_i = i\mu & \forall\ i \end{cases}$$

- On pose $\rho = \lambda/\mu$, les probabilités stationnaires sont données par :

$$\pi_k = \pi_0 \frac{\rho^k}{k!}, \quad \pi_0 = e^{-\rho}$$

- Nombre moyen de clients $E(N) = \sum_{k=1}^{\infty} k\pi_k = \rho$
- Le temps moyen de réponse est égal au temps moyen de service $1/\mu$
- On peut retrouver ce résultat par la formule de Little :
  $E(R) = \frac{E(N)}{\lambda} = \frac{\rho}{\lambda} = \frac{1}{\mu}$

Exemple : Centre de Calcul

- Les étudiants se rendent au centre de calcul selon un processus de Poisson avec un taux de 10 personnes par heure
- Chaque étudiant travaille $20mn$ en moyenne. La distribution du temps de travail est exponentielle
- Il y a 5 terminaux
- Les étudiants disent qu'ils attendent trop.....

Analyse du centre par une M/M/5

- $\lambda = 1/6$ etudiant/minute
- $\mu = 1/20$
- $\rho = 0.666$
- Utilisation moyenne d'un terminal = 0.6666
- la probabilité d'attente = probabilité que tous les terminaux soient occupés = $\gamma = \pi_0 \frac{(m*\rho)^m}{m!(1-\rho)} = 0.33$
- Nombre moyen d'étudiants dans le centre = $m\rho + \frac{\rho\gamma}{1-\rho} = 4.0$

- Nombre moyen d'étudiants en attente $= \frac{\rho\gamma}{1-\rho} = 0.65$
- Temps moyen de séjour dans le centre $= 1/\mu(1 + \frac{\gamma}{m(1-\rho)}) = 24mn$
- Qui se décomposent en 20 minutes de travail et 4 minutes d'attente...
- Mais 90-Percentile du temps d'attente = 14 minutes...(10 pourcents des étudiants doivent attendre plus de 14 minutes)

Comment répondre?

- Augmenter le nombre de terminaux ?
- Distribuer les terminaux (sans en augmenter le nombre) ?

Augmenter le nombre de terminaux

- Combien de terminaux pour que le temps d'attente soit plus petit que 2 minutes en moyenne et plus petit que 5 minutes dans 90 pourcents des cas ?
- On augmente de 1 terminal. (m=6)
- ρ devient 0.556, $\gamma = 0.15$,
- Temps moyen d'attente = 1.1 minute
- 90 percentile = 3 minutes

Répartition

- On distribue les 5 terminaux en 5 places distinctes, avec 5 files d'attente séparées et pas d'informations globales pour choisir.
- On a donc 5 M/M/1
- Le taux de service reste identique
- Le taux d'arrivées est divisée par 5 (répartition équiprobable)

- La charge ρ reste la même
- Le temps moyen de séjour est $\frac{1/\mu}{1-\rho} = 60mn$
- La variance du temps de séjour est maintenant de 3600 (elle était de 479 dans le système centralisé)
- Très mauvaise solution : le temps moyen de séjour augmente considérablement (moyenne comme variance)
- En première analyse, un système centralisé est plus efficace pour l'utilisation des ressources et le temps d'attente
- La différence provient des états où certaiens files sont vides alors que des clients attendent dans d'autres
- Mais d'autres points peuvent modifier cette analyse

Les réseaux de files d'attente

Un Réseau de Files d'Attente (RFA) est un ensemble de files simples interconnectées. Il est caractérisé par :

- le processus d'arrivées des clients dans le système (ex. Poisson de taux λ)
- pour chaque file d'attente individuelle :
  - la distribution du temps de service (ex. Exponentielle de taux μ_i)
  - le nombre de serveurs m_i
  - la capacité de la file K_i (éventuellement infinie)
  - la discipline de service (FIFO, LIFO, etc, ..., éventuellement avec priorité préemptive/ non préemptive, ...)
- le routage des clients dans le système. On distingue plusieurs types de routage dont :

- le routage dynamique : fonction de l'état courant du système (par ex. vers la file la moins chargée)
- le routage déterministe: chaque type de client suit une route prédéterminée
- le routage probabiliste (le plus souvent considéré). Il est caractérisé par les probabilités
  - p_{0i} : la probabilité qu'un client arrivant de l'extérieur se dirige vers la file i
  - p_{ij} : la probabilité de se diriger vers la file j à la fin d'un service dans la file i
  - p_{i0} : la probabilité de se diriger vers l'extérieur à la fin d'un service dans la file i

Réseaux monoclasses/ Réseaux multiclasses

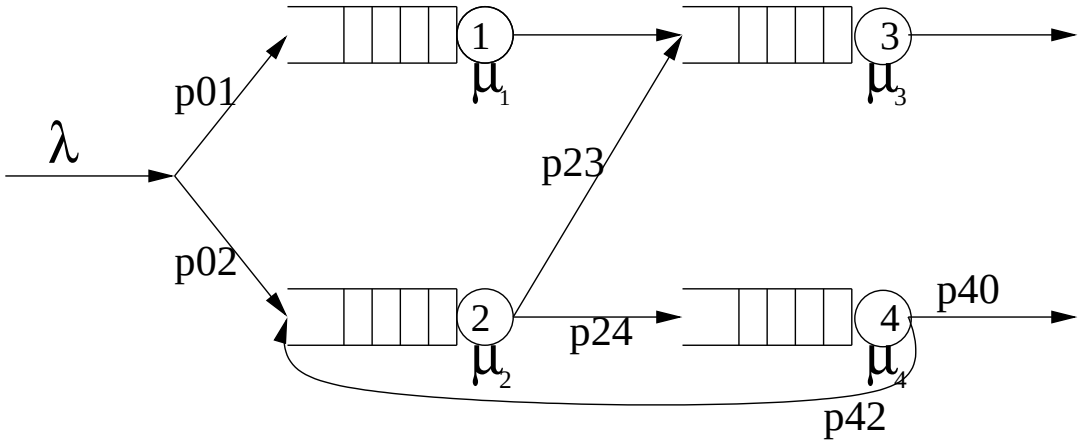
On distingue :

- les réseaux monoclasses : parcourus par un seul type de clients
- les réseaux multiclasses : où circulent plusieurs classes de clients avec des niveaux de priorité différents. Ces classes seront caractérisées par :
  - des processus d'arrivées différents
  - des routages différents
  - des temps de service différents dans chaque file

Réseaux ouverts/ Réseaux fermés

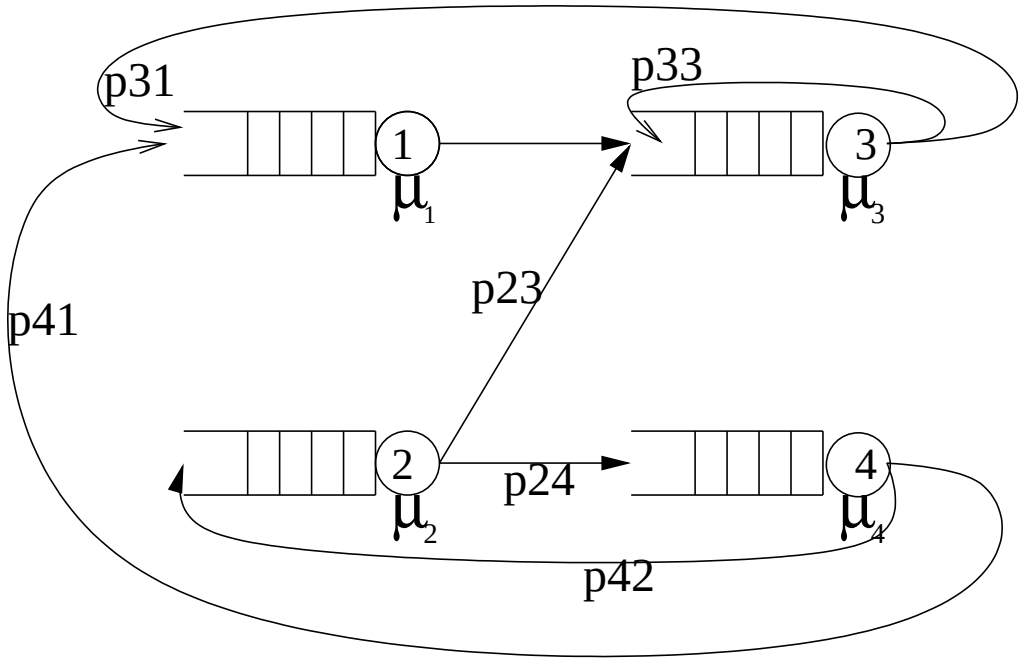
Dans un réseau ouvert :

- on peut avoir des arrivées de l'extérieur
- on peut avoir des départs vers l'extérieur
- le nombre de clients dans le réseau est variable



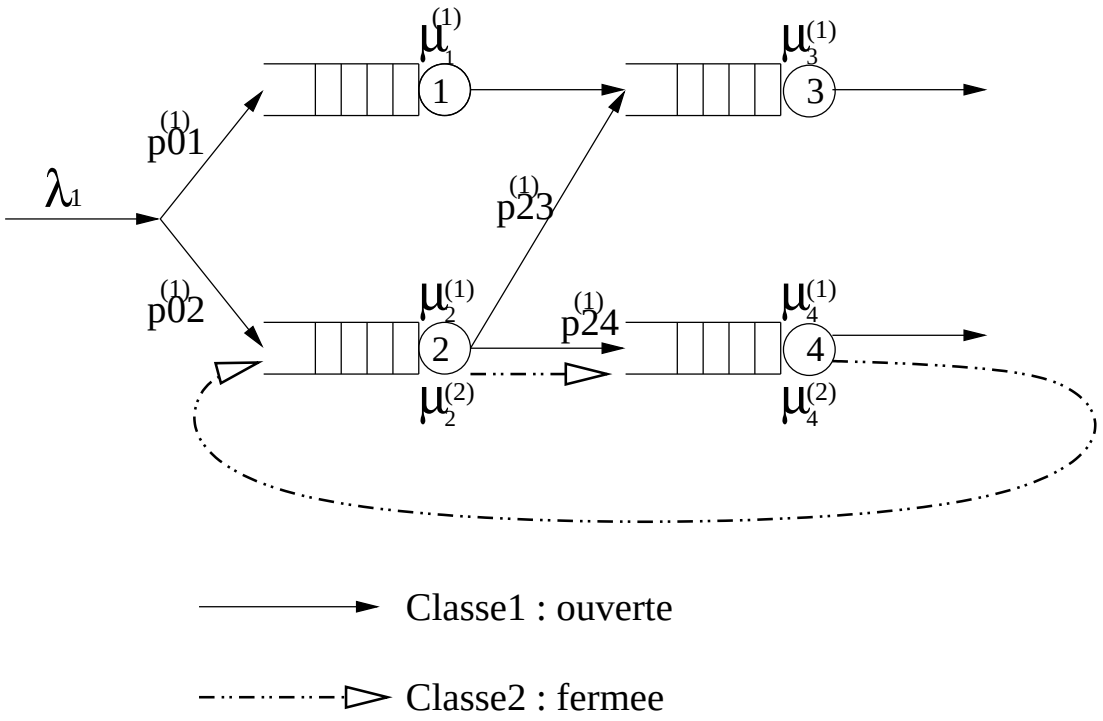
Dans un réseau fermé :

- pas d'arrivées de l'extérieur ($p_{0i} = 0$)
- pas de départs vers l'extérieur ($p_{i0} = 0$)
- le nombre de clients dans le réseau est constant



Pour les réseaux multiclasses on peut avoir des **réseaux mixtes**

- ouverts pour certaines classes
- fermés pour d'autres



Notations

Dans la suite, on se restreint aux réseaux monoclasses.

- N : le nombre de files dans le réseau
- K : le nombre constant de clients pour un réseau fermé
- $(k_1, k_2, \ldots, k_N)$: l'état du réseau, k_i étant le nombre de clients dans la file i ($\sum_{i=1}^{N} k_i = K$ pour un réseau fermé)
- m_i : le nombre de serveurs dans la file i
- μ_i : le taux de service des clients dans la file i
- λ_{0i} : le taux d'arrivée des clients de l'extérieur vers la file i
- λ : le taux total d'arrivée des clients de l'extérieur vers le réseau ouvert ($\lambda = \sum_{i=1}^{N} \lambda_{0i}$)
- λ_i : le taux total d'arrivée des clients vers la file i
- V_i : nombre moyen de visites (ou le taux de visite) à la file i ($V_i = \lambda_i/\lambda$)

Remarque : à l'état d'équilibre le taux de sortie d'une file est égal au taux d'arrivée, ainsi

- Pour un réseau ouvert :

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{N} \lambda_j p_{ji}$$

$$V_i = p_{0i} + \sum_{j=1}^{N} V_j p_{ji}$$

- Pour un réseau fermé :

$$\lambda_i = \sum_{j=1}^{N} \lambda_j p_{ji}$$

$$V_i = \sum_{j=1}^{N} V_j p_{ji}$$

Mesures de performance

Calculer la distribution stationnaire jointe $\pi(k_1, k_2, \ldots, k_N)$ est le principal problème pour les réseaux de files d'attente. En effet, les mesures de performance des files se déduisent à partir de cette distribution :

- la distribution stationnaire de la file i :
  $\pi_i(k) = \sum_{k_i=k} \pi(k_1, k_2, \ldots, k_N)$
- l'utilisation ρ_i de la file i : $\rho_i = \sum_{k=1}^{\infty} \pi_i(k) = 1 - \pi_i(0)$ (pour un réseau fermé $\pi_i(k) = 0, \quad \forall k > K$)
- le nombre moyen de clients dans la file i : $Q_i = \sum_{k=1}^{\infty} k\pi_i(k)$
- R_i : temps moyen de réponse pour la file i : $R_i = Q_i/\lambda_i$
- ...

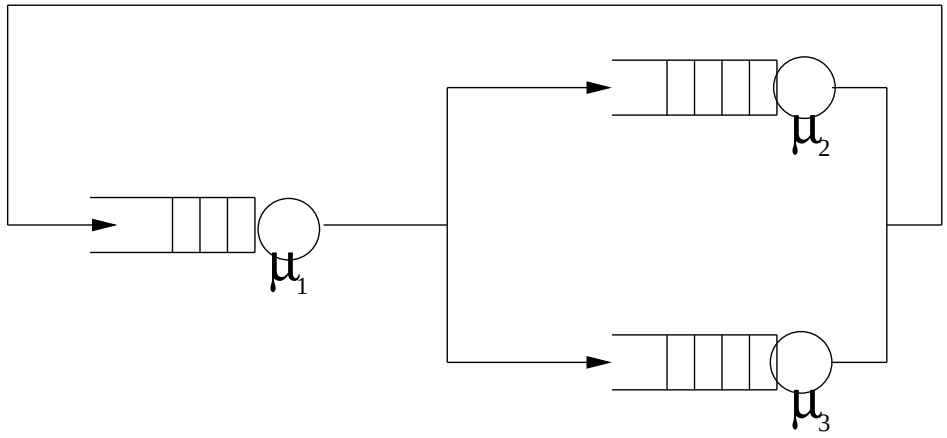
Calcul de la distribution stationnaire jointe π

- Le comportement de plusieurs réseaux de files d'attentes peut être décrit en utilisant des CTMC
- Le calcul de π peut alors se faire par des techniques numériques qui permettent de résoudre le système $\pi Q = 0$, $\pi e = 1$ (équations de balance globale) où Q est le générateur de la CTMC
- La complexité élevée de ces techniques limite leurs utilisation pour des grands réseaux, d'où la nécessité de solutions alternatives
- Une large partie de RFA appelée *réseaux à forme produit* (ayant certaines hypothèses sur les distributions des inter-arrivées et de service) possède une solution analytique simple pour le calcul de π
- Pour ces réseaux les équations de balance globale se décomposent en *équations de balance locale* et π s'exprime comme produit des probabilités stationnaires des files

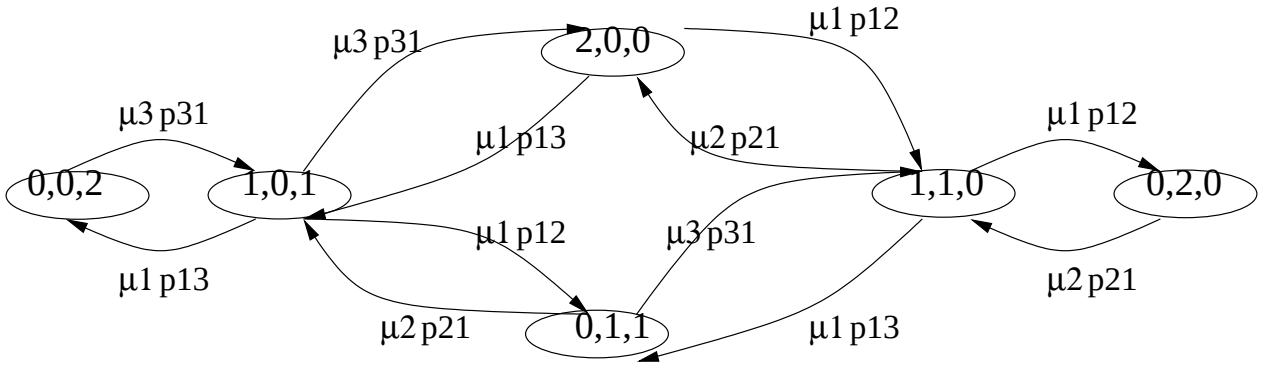
Notion de balance locale

Propriété de balance locale pour un noeud i : Le taux de départ d'un état du réseau dû au départ d'un client de la file i est égal au taux d'arrivée à cet état dû à l'arrivée d'un client à cette file

Exemple : considérons le réseau fermé du schéma avec : $K = 2$, des durées de service exponentielles



ce réseau peut être modélisé par une CMTC à six états



Les équations de balance globale s'écrivent:

(1) $\pi(2,0,0)(\mu_1 p_{12} + \mu_1 p_{13}) = \pi(1,0,1)\mu_3 p_{31} + \pi(1,1,0)\mu_2 p_{21}$

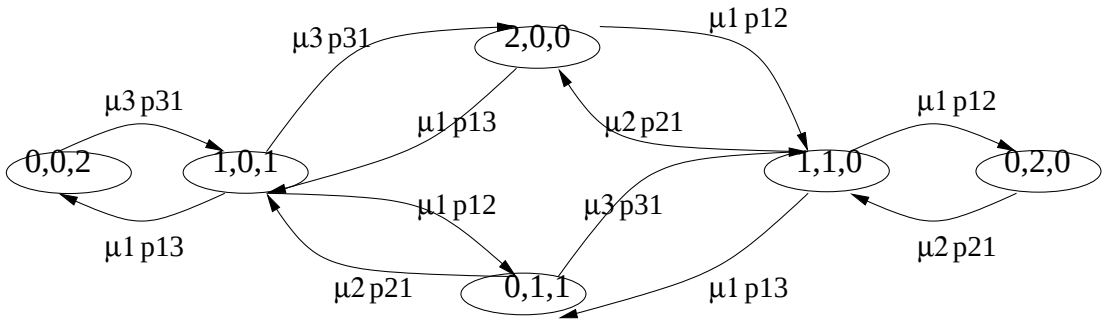
(2) $\pi(0,2,0)\mu_2 p_{21} = \pi(1,1,0)\mu_1 p_{12}$

(3) $\pi(0,0,2)\mu_3 p_{31} = \pi(1,0,1)\mu_1 p_{13}$

(4) $\pi(1,1,0)(\mu_2 p_{21} + \mu_1 p_{13} + \mu_1 p_{12}) =$
$\pi(0,2,0)\mu_2 p_{21} + \pi(2,0,0)\mu_1 p_{12} + \pi(0,1,1)\mu_3 p_{31}$

(5) $\pi(1,0,1)(\mu_3 p_{31} + \mu_1 p_{12} + \mu_1 p_{13}) =$
$\pi(0,0,2)\mu_3 p_{31} + \pi(0,1,1)\mu_2 p_{21} + \pi(2,0,0)\mu_1 p_{13}$

(6) $\pi(0,1,1)(\mu_3 p_{31} + \mu_2 p_{21}) = \pi(1,1,0)\mu_1 p_{13} + \pi(1,0,1)\mu_1 p_{12}$



équations de balance locale pour l'état $(1, 1, 0)$:

(4.1) $\pi(1,1,0)\mu_2 p_{21} = \pi(2,0,0)\mu_1 p_{12}$

le taux de départ de $(1, 1, 0)$ dû au départ d'un client de la file 2 (partie gauche) est égal au taux d'arrivée à cet état dû à l'arrivée d'un client à la file 2 (partie droite)

(4.2) $\pi(1,1,0)(\mu_1 p_{13} + \mu_1 p_{12}) = \pi(0,2,0)\mu_2 p_{21} + \pi(0,1,1)\mu_3 p_{31}$

(4.1)+ **(4.2)**=**(4)**, on a aussi :

(5.1) $\pi(1,0,1)(\mu_1 p_{12} + \mu_1 p_{13}) = \pi(0,0,2)\mu_3 p_{31} + \pi(0,1,1)\mu_2 p_{21}$

(5.2) $\pi(1,0,1)\mu_3 p_{31} = \pi(2,0,0)\mu_1 p_{13}$

(6.1) $\pi(0,1,1)\mu_2 p_{21} = \pi(1,0,1)\mu_1 p_{12}$

(6.2) $\pi(0,1,1)\mu_3 p_{31} = \pi(1,1,0)\mu_1 p_{13}$

La résolution est plus facile avec ces équations locales

Balance locale et forme produit

- Remarque : contrairement au balance globale, la propriété de balance locale n'est pas toujours vérifiée (l'équilibre locale est une condition suffisante pour l'équilibre globale mais pas nécessaire)
- lorsqu'il existe une solution pour les équations de balance locale, on dit que le réseau possède la propriété de balance locale, cette solution est l'unique solution des équations de balance globale
- Si chaque noeud du réseau vérifie la propriété de balance locale, alors :
  - tout le réseau possède la propriété de balance locale
  - il existe une solution à forme produit pour le réseau : $\pi(k_1, k_2, \ldots, k_N) = \frac{1}{G}\Pi_{i=1}^{N} f_i(k_i)$, où G est une constante de normalisation et $f_i(k_i)$ est une fonction de $\pi_i(k_i)$

Les réseaux de Jackson

- réseaux monoclasses ouverts
- arrivées Poissonniens de l'extérieur vers les noeuds du réseau
- services exponentiellement distribués dans tous les noeuds
- la discipline de service est FIFO dans tous les noeuds
- routage probabiliste
- un seul serveur de taux μ_i dans chaque noeud i

Théorème : Sous la condition de stabilité $\lambda_i < \mu_i, \forall\ i = 1, \ldots, N$, la probabilité stationnaire du réseau possède la forme produit suivante : $\pi(k_1, k_2, \ldots, k_N) = \Pi_{i=1}^{N} \pi_i(k_i)$, où $\pi_i(k_i)$ est la probabilité stationnaire d'une file M/M/1, soit $\pi_i(k_i) = (1 - \rho_i)\rho_i^{k_i}$ avec $\rho_i = \frac{\lambda_i}{\mu_i}$

Preuve

- Le processus $X(t) = (k_1(t), \ldots, k_N(t))$ est une CMTC
- Les équations de balance globale s'écrivent :

  $\pi(k_1, \ldots, k_N)(\lambda + \sum_{i=1}^{N} \mu_i \ 1_{k_i>0})$
  $= \sum_{i=1}^{N} \pi(k_1, \ldots, k_i - 1, \ldots, k_N)\lambda \ p_{0i} \ 1_{k_i>0}$
  $+ \sum_{i=1}^{N} \pi(k_1, \ldots, k_i + 1, \ldots, k_N)\mu_i \ p_{i0}$
  $+ \sum_{i=1}^{N} \sum_{j=1}^{N} \pi(k_1, \ldots, k_j + 1, \ldots, k_i - 1, \ldots, k_N)\mu_j \ p_{ji} \ 1_{k_i>0}$
- La solution d'une CMTC étant unique, il suffit de vérifier que la solution à forme produit est bien une solution de ces équations
- En divisant les deux parts de l'équation par $\pi(k_1, \ldots, k_N)$ et en remplacant les probabilités par leurs valeurs, il suffit de vérifier :

  $$\lambda + \sum_{i=1}^{N} \mu_i 1_{k_i>0} = \sum_{i=1}^{N} \frac{\lambda p_{0i} \ 1_{k_i>0}}{\rho_i} + \sum_{i=1}^{N} \rho_i \mu_i \ p_{i0} + \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\rho_j \mu_j p_{ji} 1_{k_i>0}}{\rho_i}$$

  (rappel : $\lambda_i = \rho_i \mu_i$)

- $\lambda_i = \lambda V_i$ est le débit d'entrée à la station i. Le système étant stable, c'est aussi le débit de sortie de cette station. $\lambda_i p_{i0}$ est donc le débit des clients qui quittent le système par la station i
- ainsi, $\sum_{i=1}^{N} \lambda_i \; p_{i0} = \lambda$ est bien vérifiée (débit de sortie du système= débit d'entrée dans le système)
- $\sum_{i=1}^{N} \frac{\lambda p_{0i} \; 1_{k_i>0}}{\rho_i} + \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\rho_j \mu_j p_{ji} 1_{k_i>0}}{\rho_i} =$
  $\sum_{i=1}^{N} \frac{1_{k_i>0}}{\rho_i} \; [\lambda p_{0i} + \sum_{j=1}^{N} \lambda V_j p_{ji}] = \sum_{i=1}^{N} \frac{1_{k_i>0}}{\rho_i} \; \lambda V_i = \sum_{i=1}^{N} \mu_i 1_{k_i>0}$

Remarques

- le théorème de Jackson est plus général, il inclut le cas :
  - des files avec des taux d'arrivées et de service dépendant de l'état k_i des clients dans la file i
  - des files multiserveurs sous la condition de stabilité $\lambda_i < m_i \mu_i$ (cas particulier du précédent cas en prenant $\mu_i(k_i) = \inf(k_i, m_i)\mu_i$)
- un réseau de Jackson se comporte comme un ensemble de files M/M/1 considérées en isolation
- cependant, les flux d'arrivées à l'intérieur du réseau ne sont pas nécessairement poissonniens (par exemple en cas de rebouclages), d'où l'importance du théorème

D'autres réseaux à forme produit

- **réseaux de Gordon et Newell :** réseaux monoclasses fermés
- **réseaux BCMP :** étendent les résultats de Jackson et Gordon/Newell aux réseaux de files d'attente multiclasses pouvant être ouverts, fermés, mixtes en considérant des disciplines de service différentes et des distributions de temps de service générales