




Communications dans les clusters


Olivier GLÜCK
 Université LYON 1/UFR d'Informatique
 ENS LIP projet INRIA RESO
 Olivier.Gluck@ens-lyon.fr
<http://www710.univ-lyon1.fr/~ogluck>

Copyright

- Copyright © 2006 Olivier Glück; all rights reserved
- Ce support de cours est soumis aux droits d'auteur et n'est donc pas dans le domaine public. Sa reproduction est cependant autorisée à condition de respecter les conditions suivantes :
 - Si ce document est reproduit pour les besoins personnels du reproducteur, toute forme de reproduction (totale ou partielle) est autorisée à la condition de citer l'auteur.
 - Si ce document est reproduit dans le but d'être distribué à des tierces personnes, il devra être reproduit dans son intégralité sans aucune modification. Cette notice de copyright devra donc être présente. De plus, il ne devra pas être vendu.
 - Cependant, dans le seul cas d'un enseignement gratuit, une participation aux frais de reproduction pourra être demandée, mais elle ne pourra être supérieure au prix du papier et de l'encre composant le document.
 - Toute reproduction sortant du cadre précisé ci-dessus est interdite sans accord préalable écrit de l'auteur.


Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 2



Plan

- Introduction
 - Modèle client/serveur, API socket et RPC
 - Communications inter-processus bloquantes/non bloquantes...
- Clusters et communications
 - Architecture et composants d'un cluster
 - Modèles à mémoire partagée/distribuée
 - La bibliothèque de communication MPI
- Problèmes liés à la communication dans les clusters
 - Optimisations des communications
 - Les réseaux rapides dans les clusters (Myrinet, Quadrics, Infiniband, ...)
 - Application à l'implantation de MPI sur un réseau rapide disposant d'une primitive d'écriture distante


Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 3



Bibliographie


- « La communication sous Unix », 2ième édition, Jean-Marie Rifflet, Ediscience international, ISBN 2-84074-106-7
- « *Bulding Clustered Linux Systems* », R. W. Lucke, Prentice Hall PTR, ISBN 0-13-144853-6
- « *High Performance Cluster Computing* », Vol. 1&2, R. Buyya, Prentice Hall PTR, ISBN 0-13-013784-7
- « *Beowulf Cluster Computing with Linux* », 2nd edition, Gropp, M.I.T. Press, ISBN 0-262-69292-9
- « *Operating Systems : Internals and Design Principles* », 5th Edition, W. STALLINGS, Pearson Education International, ISBN 0-13-127837-1
- Internet...
 - <http://www.cs.mu.oz.au/678/>
 - <http://www.cs.wmich.edu/~gupta/teaching/cs526/sp03/tutorials/mpich.html>
 - <http://www.buyya.com/>
 - http://www.cs.wmich.edu/gupta/teaching/cs626/w98/mpich_doc.html

Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 4



Introduction

Modèle client/serveur
 Communications inter-processus
 Rappel : API socket et RPC

Les applications réseau (1)

- Applications = la raison d'être des réseaux infos
- Profusion d'applications depuis 30 ans grâce à l'expansion d'Internet
 - années 1980/1990 : les applications "textuelles"
 - messagerie électronique, accès à des terminaux distants, transfert de fichiers, groupe de discussion (forum, *newsgroup*), dialogue interactif en ligne (chat), la navigation Web
 - plus récemment :
 - les applications multimédias : vidéo à la demande (*streaming*), visioconférences, radio et téléphonie sur Internet
 - la messagerie instantanée (ICQ, MSN Messenger)
 - les applications *Peer-to-Peer* (MP3, ...)

Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 6

Les applications réseau (2)

- L'application est généralement répartie (ou distribuée) sur plusieurs systèmes
- Exemples :
 - L'application Web est constituée de deux logiciels communicants : le navigateur client qui effectue une requête pour disposer d'un document présent sur le serveur Web
 - L'application *telnet* : un terminal virtuel sur le client, un serveur *telnet* distant qui exécute les commandes
 - La visioconférence : autant de clients que de participants
- --> Nécessité de disposer d'un protocole de communication applicatif !

Terminologie des applications réseau

- Processus :
 - une entité communicante
 - un programme qui s'exécute sur un hôte d'extrémité
- Communications inter-processus locales :
 - communications entre des processus qui s'exécutent sur un même hôte
 - communications régies par le système d'exploitation (tubes UNIX, mémoire partagée, ...)
- Communications inter-processus distantes :
 - les processus s'échangent des **messages** à travers le réseau selon un **protocole** de la couche applications
 - nécessite une infrastructure de transport sous-jacente

Protocoles de la couche Applications

- Le protocole applicatif définit :
 - le format des messages échangés entre les processus émetteur et récepteur
 - les types de messages : requête, réponse, ...
 - l'ordre d'envoi des messages
- Exemples de protocoles applicatifs :
 - HTTP pour le Web, POP/IMAP/SMTP pour le courrier électronique, SNMP pour l'administration de réseau, ...
- Ne pas confondre le protocole et l'application !
 - Application Web : un format de documents (HTML), un navigateur Web, un serveur Web à qui on demande un document, un protocole (HTTP)

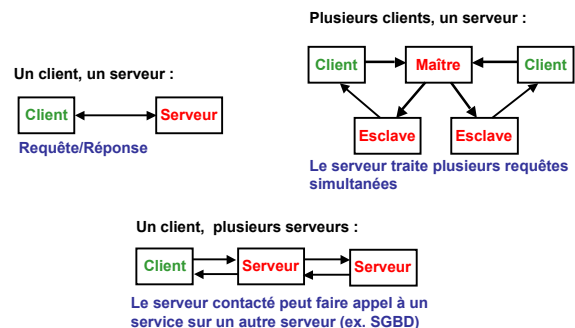
Le modèle Client / Serveur

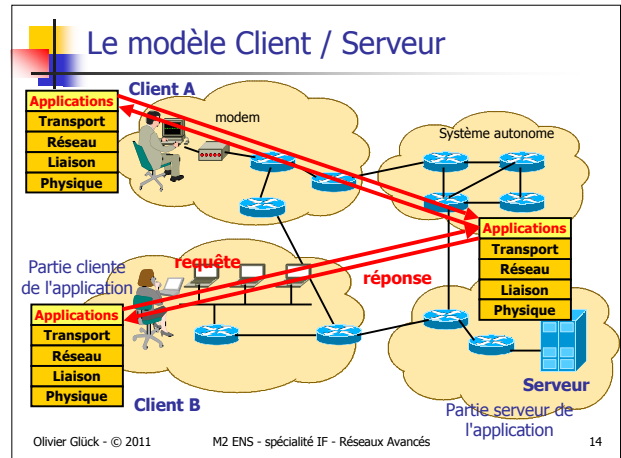
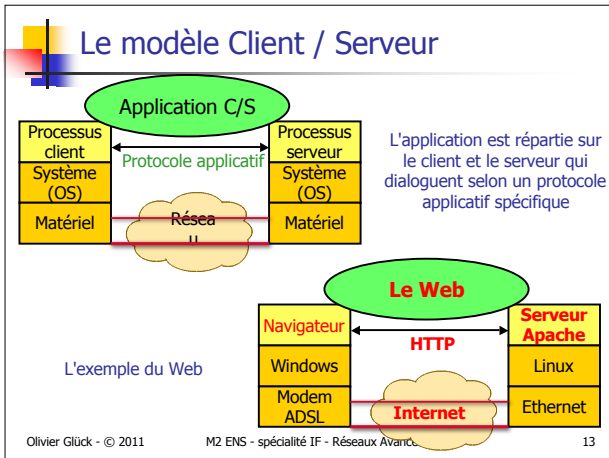
- Idée : l'application est répartie sur différents sites pour optimiser le traitement, le stockage...
- Le client
 - effectue une demande de service auprès du serveur (**requête**)
 - initie le contact (parle en premier), ouvre la session
- Le serveur
 - est la partie de l'application qui offre un service
 - est à l'écoute des requêtes clientes
 - répond au service demandé par le client (**réponse**)

Le modèle Client / Serveur

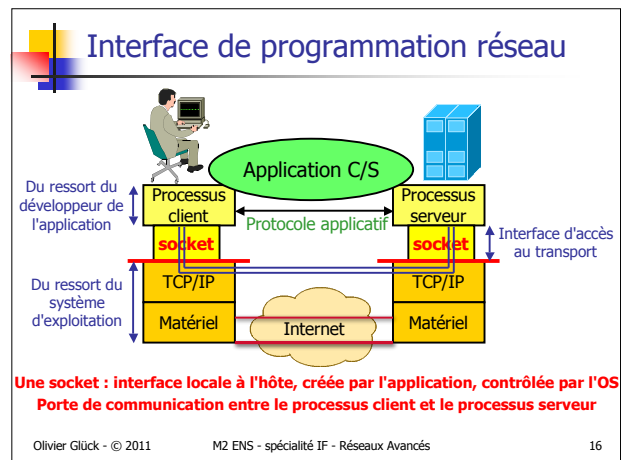
- Le client et le serveur ne sont pas identiques, ils forment un système coopératif
 - les parties client et serveur de l'application peuvent s'exécuter sur des systèmes différents
 - une même machine peut implanter les côtés client ET serveur de l'application
 - un serveur peut répondre à plusieurs clients simultanément

Des clients et des serveurs...

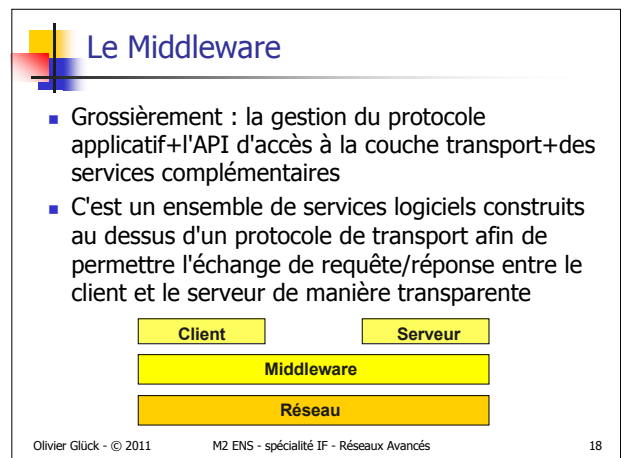




- ## Interface de programmation réseau
- Il faut une interface entre l'application réseau et la couche transport
 - le transport n'est qu'un tuyau (TCP ou UDP dans Internet)
 - l'API (*Application Programming Interface*) n'est que le moyen d'y accéder (interface de programmation)
 - Les principales APIs de l'Internet
 - les sockets
 - apparus dans UNIX BSD 4.2
 - devenus le standard de fait
 - les RPC : Remote Procedure Call - appel de procédures distantes
- Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 15



- ## Application C/S - récapitulatif
- Une application Client/Serveur, c'est
 - une partie cliente** qui exécute des requêtes vers un serveur
 - une partie serveur** qui traite les requêtes clientes et y répond
 - un protocole applicatif** qui définit les échanges entre un client et un serveur
 - un accès via une API** (interface de programmation) à la couche de transport des messages
 - Bien souvent les parties cliente et serveur ne sont pas écrites par les mêmes programmeurs (Navigateur Netscape/Serveur apache) --> rôle important des RFCs qui spécifient le protocole !
- Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 17



Le Middleware

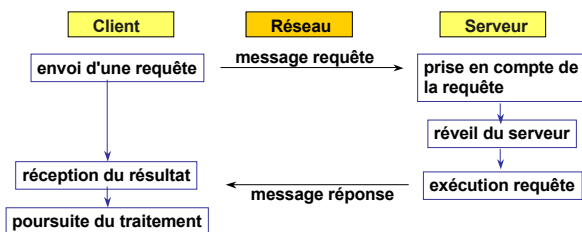
- Complément de services du réseau permettant la réalisation du dialogue client/serveur :
 - prend en compte les requêtes de l'application cliente
 - les transmet de manière transparente à travers le réseau jusqu'au serveur
 - prend en compte les données résultat du serveur et les transmet vers l'application cliente
- L'objectif essentiel du *middleware* est d'offrir aux applications une interface unifiée permettant l'accès à l'ensemble des services disponibles sur le réseau : l'API

Fonctions d'un Middleware

- Procédures d'établissement/fermeture de connexion
- Exécution des requêtes, récupération des résultats
- Initiation des processus sur différents sites
- Services de répertoire
- Accès aux données à distance
- Gestion d'accès concurrents
- Sécurité et intégrité (authentification, cryptage, ...)
- Monitoring (compteurs, ...)
- Terminaison de processus
- Mise en cache des résultats, des requêtes

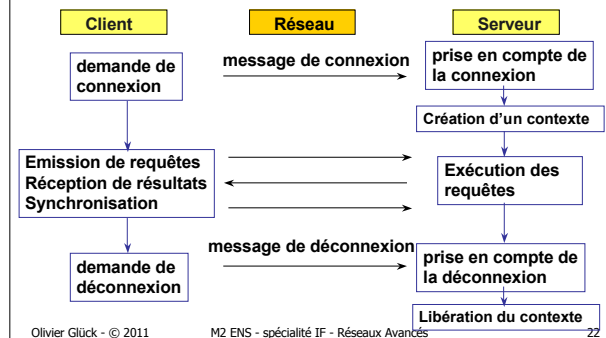
Les modes de communication

- Communication en mode non connecté



Les modes de communication

- Communication en mode connecté

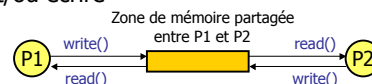


Les schémas de communication

- Dès lors qu'une application est répartie, elle se décompose en plusieurs processus qui doivent communiquer (échanges de données)
- Deux grands types de schéma de communication
 - communication par mémoire partagée (ou fichier)
 - communication par passage de messages
- On retrouve ces deux schémas de communication
 - dans des **communications locales** : entre processus s'exécutant sur le même hôte
 - dans des **communications distantes** : entre processus s'exécutant sur des hôtes distants

Communication par mémoire partagée

- Les processus se partagent une zone de mémoire commune dans laquelle ils peuvent lire et/ou écrire



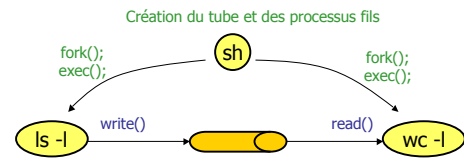
- Intérêt : communications transparentes, limitation des copies mémoire
- Problème : gestion de l'accès à une ressource partagée
 - problème si deux écritures simultanées (ordre d'ordonnement, atomicité des opérations)
 - les processus P1 et P2 doivent se synchroniser pour accéder au tampon partagé (verrou, sémaphore, ...)

Communication par mémoire partagée

- Communications locales
 - les deux processus s'exécutent sur la même machine donc peuvent se partager une partie de leur espace d'adressage
 - exemple : les *threads* s'exécutent dans le contexte d'un même processus
- Communications distantes
 - la mémoire partagée est physiquement répartie
 - le gestionnaire de mémoire virtuelle permet de regrouper les différents morceaux selon un seul espace d'adressage
 - problème de cohérence mémoire...

Les tubes de communication (*pipes*)

- Communications locales type mémoire partagée
 - le canal de communication est unidirectionnel (pas de problème de synchronisation)
 - communications entre 2 processus uniquement : l'un écrit dans le tube, l'autre lit
- Exemple : `sh$ ls -l | wc -l`



Communication par passage de msg

- Les processus n'ont pas accès à des "variables" communes
- Ils communiquent en s'échangeant des messages
 - au moins deux primitives : *send()* et *recv()*
 - des zones de mémoire locales à chaque processus permettent l'envoi et la réception des messages
 - l'émetteur/récepteur doit pouvoir désigner le récepteur/émetteur distant
- Problèmes
 - zones d'émission et réception distinctes ?
 - nombre d'émetteurs/récepteurs dans une zone ?
 - opérations bloquantes/non bloquantes ?

Communication par passage de msg

- Il faut éviter les écritures concurrentes :
- Pour se ramener à des communications point-à-point
 - --> dissocier le tampon d'émission et de réception
 - --> avoir autant de tampons de réception que d'émetteurs potentiels
 - --> il ne reste plus alors au protocole qu'à s'assurer que deux émissions successives (d'un même émetteur) n'écrasent pas des données non encore lues (contrôle de flux)

Opérations bloquantes/non bloquantes

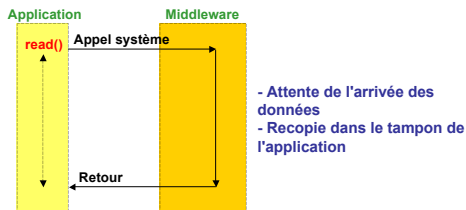
- Quand un appel à une primitive *send()* ou *recv()* doit-il se terminer ?
- Plusieurs sémantiques en réception :
 - *recv()* peut rendre la main
 - aussitôt (*recv()* non bloquant)
 - quand les données ont été reçues et copiées depuis le tampon de réception local (le tampon de réception est de nouveau libre)

Opérations bloquantes/non bloquantes

- Plusieurs sémantiques en émission :
 - *send()* peut rendre la main
 - aussitôt (*send()* non bloquant)
 - quand les données ont été copiées dans le tampon d'émission local (les données peuvent être modifiées au niveau de l'application)
 - quand les données ont été copiées dans le tampon de réception distant (le tampon d'émission local est de nouveau libre)
 - quand le destinataire a consommé les données (le tampon de réception sur le destinataire est de nouveau libre)

Opérations bloquantes

- Le processus se bloque jusqu'à ce que l'opération se termine :



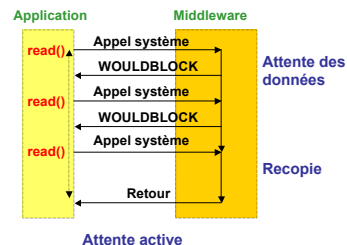
Opérations non bloquantes

- Intérêt :
 - le processus peut faire autre chose en attendant que les données soient émises ou reçues
- Le processus a tout de même besoin d'être informé de la complétion de l'opération (lecture ou écriture)
- Deux possibilités :
 - attente active : appels réguliers à la primitive jusqu'à complétion
 - attente passive : le système informe le processus par un moyen quelconque de la complétion de l'opération (signaux par exemple)

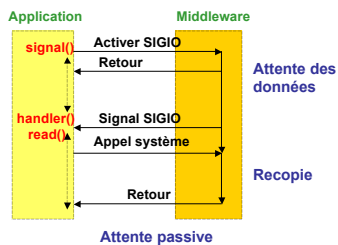
Communication par signaux

- Mécanisme de communications **locales** inter-processus (ou depuis le noyau vers un processus) permettant de notifier un événement
- Principe : interruption logicielle quand l'événement se produit
- Le processus
 - indique les signaux qu'il souhaite capter (provoquant son interruption)
 - met en place un *handler* (fonction particulière) qui sera exécuté quand l'événement se produira
- Exemple : arrivée de données urgentes sur une socket

Opérations non bloquantes



Opérations non bloquantes



Désignation du destinataire/émetteur

- Pour faire du passage de messages, il est nécessaire de désigner l'autre extrémité de la communication
- Désignation explicite
 - du ou des processus destinataire(s)/émetteurs
- Désignation implicite
 - recevoir un message de n'importe qui
 - émettre un message à n'importe qui (diffusion)
 - une phase d'établissement de connexion désigne les deux entités communicantes

Les sockets - adressage

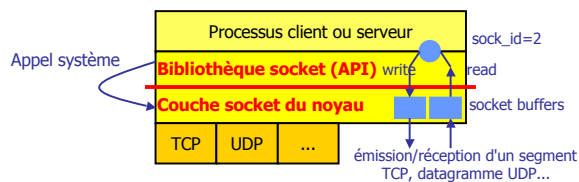
- Deux processus communiquent en émettant et recevant des données via les sockets
- Les sockets sont des portes d'entrées/sorties vers le réseau (la couche transport)
- Une socket est identifiée par une adresse de transport qui permet d'identifier les processus de l'application concernée
- Une adresse de transport = un numéro de port (identifie l'application) + une adresse IP (identifie le serveur ou l'hôte dans le réseau)

Les sockets - adressage

- Le serveur doit utiliser un numéro de port fixe vers lequel les requêtes clientes sont dirigées
- Les ports inférieurs à 1024 sont réservés :
 - "well-known ports"
 - ils permettent d'identifier les serveurs d'applications connues
 - ils sont attribués par l'IANA
- Les clients n'ont pas besoin d'utiliser des *well-known ports*
 - ils utilisent un port quelconque entre 1024 et 65535 à condition que le triplet <transport/@IP/port> soit unique
 - ils communiquent leur numéro de port au serveur lors de la requête (à l'établissement de la connexion TCP ou dans les datagrammes UDP)

Les sockets en pratique

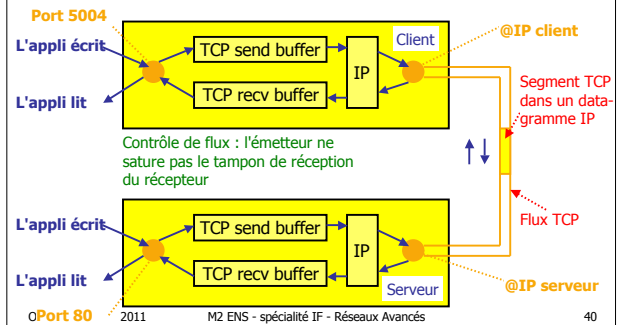
Un descripteur de socket (*sock_id*) n'est qu'un point d'entrée vers le noyau



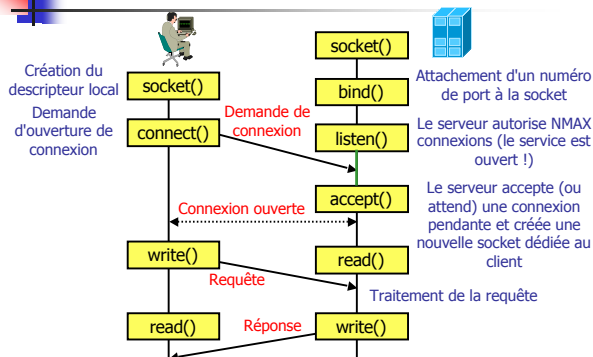
- la bibliothèque socket est liée à l'application
- la couche socket du noyau réalise l'adaptation au protocole de transport utilisé

Rappel - une connexion TCP

- Une connexion = (**@IP_src, port_src, @IP_dest, port_dest**)



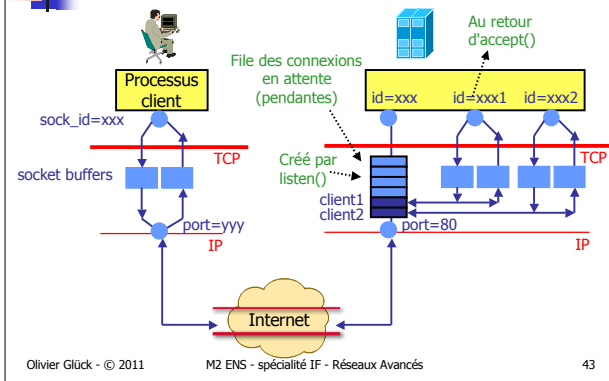
En mode connecté...



En mode connecté...

	Paramètres en entrée	Paramètres en sortie
<code>socket()</code>	type, domaine, protocole	sock_id
<code>bind()</code>	sock_id, port	
<code>listen()</code>	sock_id, NMAX	
<code>connect()</code>	sock_id, @sock_dest	
<code>accept()</code>	sock_id	@sock_src, client_sock_id
<code>read()</code>	client_sock_id, @recv_buf, lg	read_lg
<code>write()</code>	client_sock_id, @send_buf, lg	write_lg

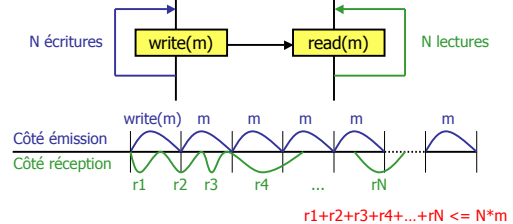
En mode connecté...



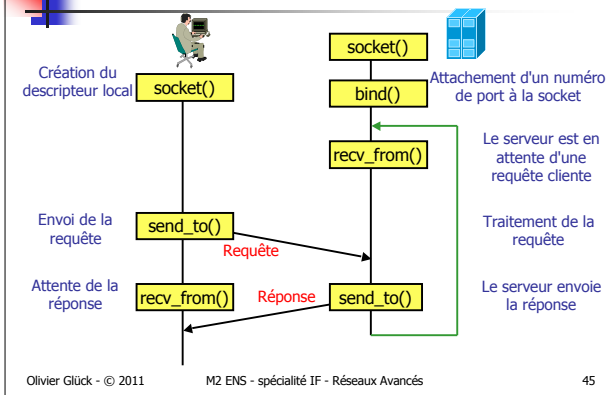
En mode connecté...

- Attention : les émissions/réceptions ne sont pas synchrones

- $read(m)$: lecture **d'au plus** m caractères
- $write(m)$: écriture de m caractères



En mode non connecté...



En mode non connecté...

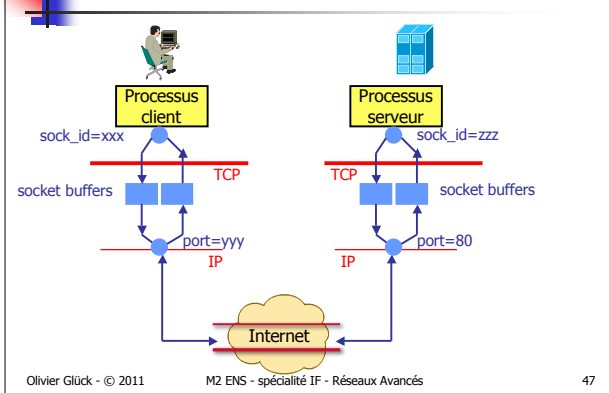
	Paramètres en entrée	Paramètres en sortie
socket()	type, domaine, protocole	sock_id
bind()	sock_id, port	
recv_from()	sock_id, @recv_buf, lg	read_lg, @sock_src
send_to()	sock_id, @sock_dest, @send_buf, lg	write_lg

Rappel en mode connecté :

read()	client_sock_id, @recv_buf, lg	read_lg
write()	client_sock_id, @send_buf, lg	write_lg

Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 46

En mode non connecté...



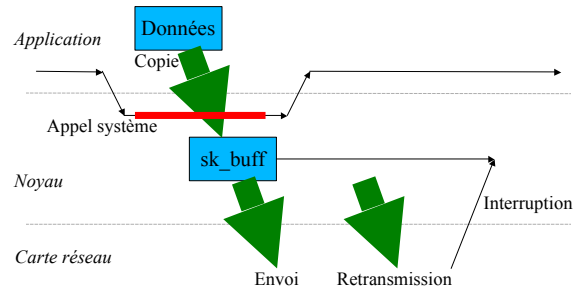
Opérations bloquantes/non bloquantes

- Par défaut, les primitives `connect()`, `accept()`, `send_to()`, `recv_from()`, `read()`, `write()` sont bloquantes
 - `recv()` sur un tampon vide attendra l'arrivée des données pour rendre la main
 - `send()` sur un tampon plein attendra que les données quittent le tampon pour rendre la main
 - `accept()` ne rend la main qu'une fois une connexion établie (bloque si pas de connexions pendantes)
 - `connect()` ne rend la main qu'une fois la connexion cliente établie (sauf si pas entre `listen()` et `accept()`)
- Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 48

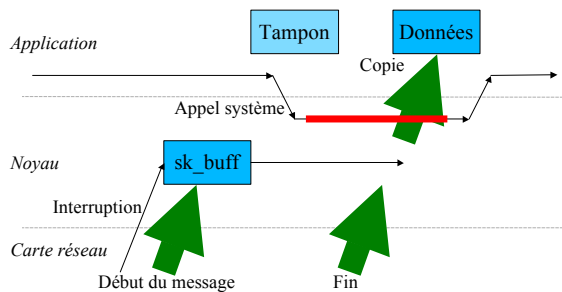
Opérations bloquantes/non bloquantes

- Il est possible de paramétrer la socket lors de sa création pour rendre les opérations non bloquantes
- Comportement d'une émission non bloquante
 - tout ce qui peut être écrit dans le tampon l'est, les caractères restants sont abandonnés (la primitive retourne le nombre de caractères écrits)
 - si aucun caractère ne peut être écrit (tampon plein), retourne -1 avec `errno=EWOULDBLOCK` (l'application doit réessayer plus tard)
- Comportement d'une lecture non bloquante
 - s'il n'y a rien à lire dans la socket, retourne -1 ... (l'application doit réessayer plus tard)

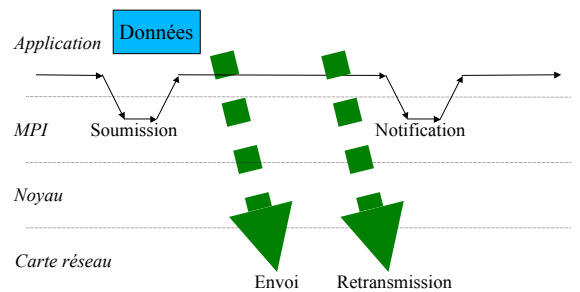
Emission bloquante



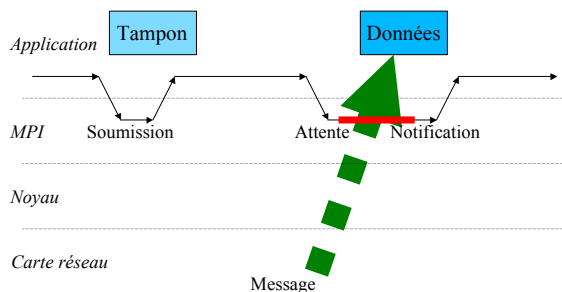
Réception bloquante



Emission non-bloquante



Réception non-bloquante



La scrutation de plusieurs sockets

- La primitive `select()` rend la main quand une de ces conditions se réalise :
 - l'un des événements attendus sur un descripteur de l'un des ensembles se réalise : les descripteurs sur lesquels l'opération est possible sont dans un paramètre de sortie
 - le temps d'attente maximum s'est écoulé
 - le processus a capté un signal (provoque la sortie de `select()`)

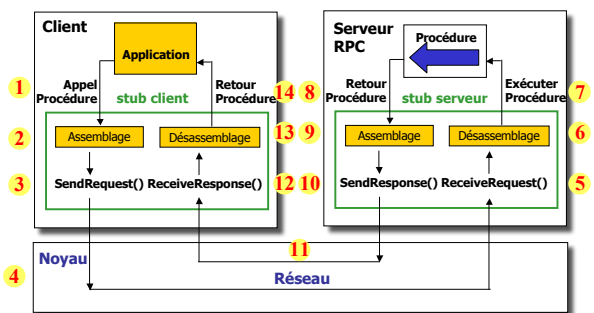
Deux approches de conception

- Un concepteur d'application distribuée peut procéder selon deux approches :
 - conception orientée communication :
 - définition du protocole d'application (format et syntaxe des messages) inter-opérant entre le client et le serveur
 - conception des composants serveur et client, en spécifiant comment ils réagissent aux messages entrants et génèrent les messages sortants
 - conception orientée application :
 - construction d'une application conventionnelle, dans un environnement mono-machine
 - subdivision de l'application en plusieurs modules qui pourront s'exécuter sur différentes machines

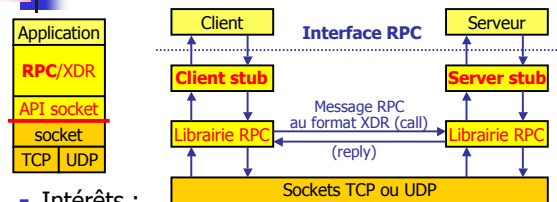
Principe général

- Souvent, quand un client envoie une requête (des paramètres), il est bloqué jusqu'à la réception d'une réponse
- Analogie avec un appel de fonction
 - la fonction ou procédure ne rend la main au programme appelant qu'une fois le traitement (calcul) terminé
- RPC - Remote Procedure Call
 - permettre à un processus de faire exécuter une fonction par un autre processus se trouvant sur une machine distante
 - se traduit par l'envoi d'un message contenant l'identification de la fonction et les paramètres
 - une fois le traitement terminé, un message retourne le résultat de la fonction à l'appelant

Le modèle RPC



L'interface RPC



- Intérêts :
 - l'application n'a pas à manipuler directement les sockets (le transport des données est transparent)
 - l'implémentation des RPC est indépendante de l'OS
- Inconvénient :
 - l'utilisation des RPC est moins performante que l'utilisation directe des sockets (couches supplémentaires)

Restrictions liées aux RPC

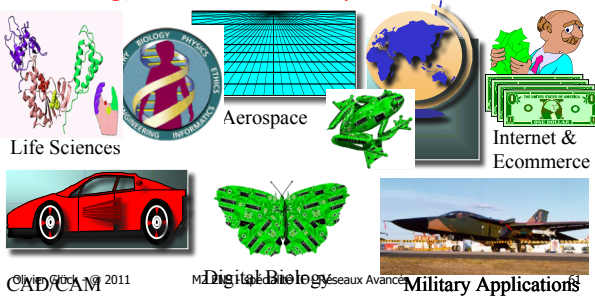
- Pas de passage de paramètres par adresse : impossible de passer des pointeurs (ou références)
 - en effet, les espaces d'adressage du client et du serveur sont différents donc aucun sens de passer une adresse
- La procédure distante n'a pas accès aux variables globales du client, aux périphériques d'E/S (affichage d'un message d'erreur !)
- Un appel de procédure obéit à fonctionnement synchrone : une instruction suivant un appel de procédure ne peut pas s'exécuter tant que la procédure appelée n'est pas terminée

High Performance Cluster Computing: Architectures and Systems
 Book Editor: Rajkumar Buyya
 Slides: Hai Jin and Raj Buyya

Internet and Cluster Computing Center
 Huazhong University of Science & Technology
 INRIA

Resource Hungry Applications

- Solving grand challenge applications using computer *modeling, simulation* and *analysis*



Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

Military Applications

How to Run Applications Faster ?

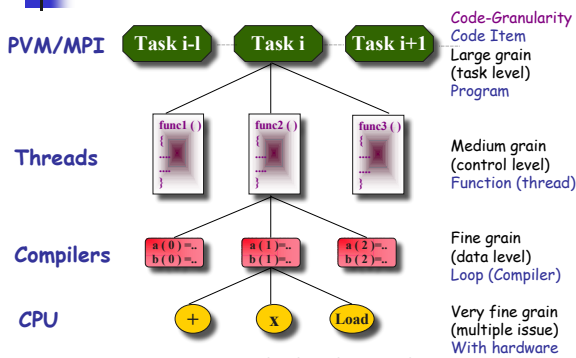
- There are 3 ways to improve performance:
 - Work Harder
 - Work Smarter
 - Get Help
- Computer Analogy
 - Using faster hardware
 - Optimized algorithms and techniques used to solve computational tasks
 - Multiple computers to solve a particular task

Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

62

Levels of Parallelism



Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

63

Scalable (Parallel) Computer Architectures

- Taxonomy
 - based on how processors, memory & interconnect are laid out, resources are managed
- Massively Parallel Processors (MPP)
- Symmetric Multiprocessors (SMP)
- Cache-Coherent Non-Uniform Memory Access (CC-NUMA)
- Clusters
- Distributed Systems – Grids/P2P

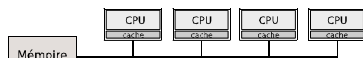
Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

64

Scalable Parallel Computer Architectures

- MPP
 - A large parallel processing system with a shared-nothing architecture
 - Consist of several hundred nodes with a high-speed interconnection network/switch
 - Each node consists of a main memory & one or more processors
 - Runs a separate copy of the OS
- SMP
 - 2-64 processors today
 - Shared-everything architecture
 - All processors share all the global resources available
 - Single copy of the OS runs on these systems



Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

65

Scalable Parallel Computer Architectures

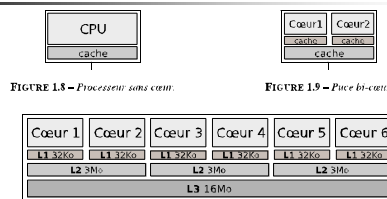
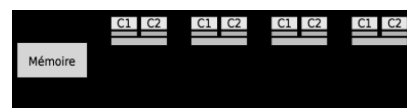


FIGURE 1.10 – Processeur hexa-cœur INTEL Xeon Dominou X7460.

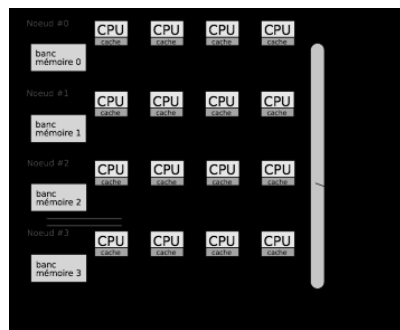


Olivier Glück - © 2011

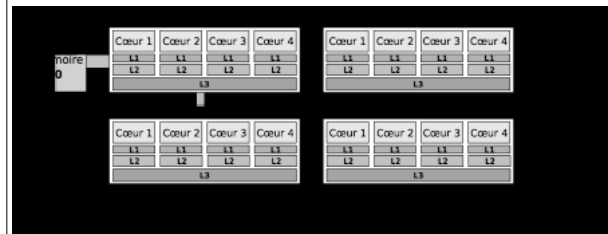
M2 ENS - spécialité IF - Réseaux Avancés

66

Scalable Parallel Computer Architectures



Scalable Parallel Computer Architectures



Scalable Parallel Computer Architectures

- CC-NUMA
 - a scalable multiprocessor system having a cache-coherent nonuniform memory access architecture
 - every processor has a global view of all of the memory
- Clusters
 - a collection of workstations / PCs that are interconnected by a high-speed network
 - work as an integrated collection of resources
 - have a single system image spanning all its nodes
- Distributed systems
 - considered conventional networks of independent computers
 - have multiple system images as each node runs its own OS
 - the individual machines could be combinations of MPPs, SMPs, clusters, & individual computers

Les supercalculateurs

- Assemblage de multiples processeurs dans une grosse machine
 - Processeurs spécifiques
 - Conçus pour collaborer avec beaucoup d'autres
 - Réseau de communication spécifique
 - Mémoire partagée
- Peu rentable
 - Peu d'exemplaires
 - Très spécifique

Les grappes de calcul ou clusters

- Utilisation de matériel standard
 - Processeurs classiques
 - Pas cher
- Ajout d'un réseau très haute performance
 - Communications assez rapides pour ne pas ralentir le calcul
- Extensif
 - Passage à l'échelle aisé

Clusters



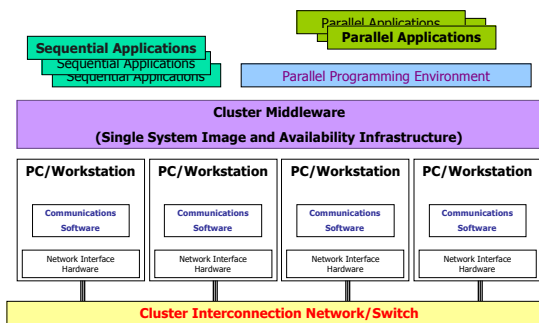
Why PC/WS Clustering Now ?

- Individual PCs/workstations are becoming increasing powerful
- Commodity networks bandwidth is increasing and latency is decreasing
- PC/Workstation clusters are easier to integrate into existing networks
- Typical low user utilization of PCs/WSS
- Development tools for PCs/WS are more mature
- PC/WS clusters are a cheap and readily available
- Clusters can be easily grown

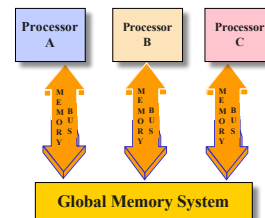
What is Cluster ?

- A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single, integrated computing resource.
- A node
 - a single or multiprocessor system with memory, I/O facilities, & OS
 - generally 2 or more computers (nodes) connected together
 - in a single cabinet, or physically separated & connected via a LAN
 - appear as a single system to users and applications
 - provide a cost-effective way to gain features and benefits

Cluster Architecture

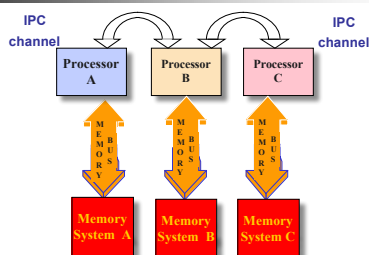


Shared Memory MIMD



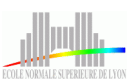


- Comm: Source PE writes data to Global Memory & destination retrieves it
- Easy to build, conventional OS of SISD can be easily be ported
 - Limitation : reliability & expandability. **A memory component or any processor failure affects the whole system.**
 - Increase of processors leads to memory contention.
Ex. : Silicon graphics supercomputers....

Distributed Memory MIMD





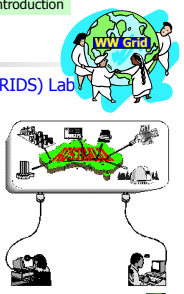
- Communication : IPC (Inter-Process Communication) via High Speed Network.
- Network can be configured to ... Tree, Mesh, Cube, etc.
- Unlike Shared MIMD
 - easily/ readily expandable
 - Highly reliable (any CPU failure does not affect the whole system)







Parallel Programming with Message-Passing Interface (MPI) An Introduction

Rajkumar Buyya

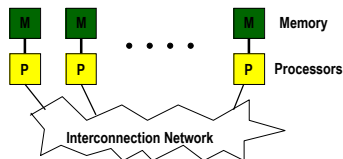
Grid Computing and Distributed Systems (GRIDS) Lab
The University of Melbourne
Melbourne, Australia
www.gridbus.org

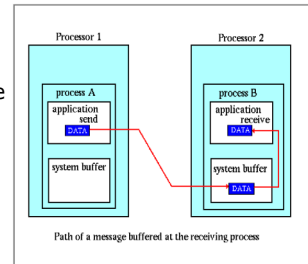
Message-Passing Programming Paradigm

- Each processor in a message-passing program runs a sub-program
 - written in a conventional sequential language
 - all variables are private
 - communicate via special subroutine calls



Messages

- Messages are packets of data moving between sub-programs
- The message passing system has to be told the following information
 - Sending processor
 - Source location
 - Data type
 - Data length
 - Receiving processor(s)
 - Destination location
 - Destination size



Messages

- Access:
 - Each sub-program needs to be connected to a message passing system
- Addressing:
 - Messages need to have addresses to be sent to
- Reception:
 - It is important that the receiving process is capable of dealing with the messages it is sent
- A message passing system is similar to:
 - Post-office, Phone line, Fax, E-mail, etc
- Message Types:
 - Point-to-Point, Collective, Synchronous (telephone) / Asynchronous (Postal)

Point-to-Point Communication

- Simplest form of message passing
- One process sends a message to another
- Several variations on how sending a message can interact with execution of the sub-program

Point-to-Point variations

- Synchronous Sends
 - provide information about the completion of the message
 - e.g. fax machines
- Asynchronous Sends
 - Only know when the message has left
 - e.g. post cards
- Blocking operations
 - only return from the call when operation has completed
- Non-blocking operations
 - return straight away - can test/wait later for completion

Collective Communications

- Collective communication routines are higher level routines involving several processes at a time
- Can be built out of point-to-point communications
- Barriers
 - synchronise processes
- Broadcast
 - one-to-many communication
- Reduction operations
 - combine data from several processes to produce a single (usually) result

Collective Communications

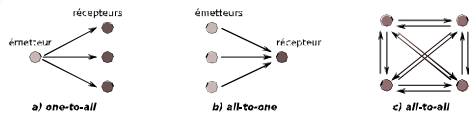


FIGURE 2.6 – Schémas de communications collectives.

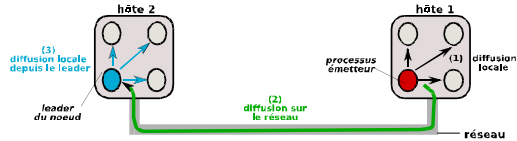


FIGURE 2.7 – Communication one-to-all sur deux nœuds disposant de 4 processus chacun.

Point-to-Point Communication

- Communication between two processes
- Source process sends message to destination process
- Communication takes place within a communicator
- Destination process is identified by its rank in the communicator
- **MPI provides four communication modes for sending messages**
 - **standard, synchronous, buffered, and ready**
- Only one mode for receiving

Standard Send

- Completes once the message has been sent
 - Note: it may or may not have been received
- Programs should obey the following rules:
 - It should not assume the send will complete before the receive begins - can lead to deadlock
 - It should not assume the send will complete after the receive begins - can lead to non-determinism
 - processes should be eager readers - they should guarantee to receive all messages sent to them - else network overload
- Can be implemented as either a buffered send or synchronous send

Standard Send

```

MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest,
         int tag, MPI_Comm comm)
buf       the address of the data to be sent
count     the number of elements of datatype buf contains
datatype  the MPI datatype
dest      rank of destination in communicator comm
tag       a marker used to distinguish different message types
comm      the communicator shared by sender and receiver
ierror    the fortran return value of the send
    
```

Standard/Blocked Send/Receive

MPI_Send

Basic blocking send operation. Routine returns only after the application buffer in the sending task is free for reuse. Note that this routine may be implemented differently on different systems. The MPI standard permits the use of a system buffer but does not require it. Some implementations may actually use a synchronous send (discussed below) to implement the basic blocking send.

```

MPI_Send (*buf, count, datatype, dest, tag, comm)
MPI_SEND (buf, count, datatype, dest, tag, comm, ierr)
    
```

MPI_Recv

Receive a message and block until the requested data is available in the application buffer in the receiving task.

```

MPI_Recv (*buf, count, datatype, source, tag, comm, *status)
MPI_RECV (buf, count, datatype, source, tag, comm, status, ierr)
    
```

MPI_Ssend

Synchronous blocking send: Send a message and block until the application buffer in the sending task is free for reuse and the destination process has started to receive the message.

```

MPI_Ssend (*buf, count, datatype, dest, tag, comm, ierr)
MPI_SSEND (buf, count, datatype, dest, tag, comm, ierr)
    
```

Non Blocking Message Passing

MPI_Isend

Identifies an area in memory to serve as a send buffer. Processing continues immediately without waiting for the message to be copied out from the application buffer. A communication request handle is returned for handling the pending message status. The program should not modify the application buffer until subsequent calls to MPI_Wait or MPI_Test indicates that the non-blocking send has completed.

```


MPI_Isend (*buf, count, datatype, dest, tag, comm, *request)
MPI_ISEND (buf, count, datatype, dest, tag, comm, request, ierr)
    
```

MPI_Irecv

Identifies an area in memory to serve as a receive buffer. Processing continues immediately without actually waiting for the message to be received and copied into the application buffer. A communication request handle is returned for handling the pending message status. The program must use calls to MPI_Wait or MPI_Test to determine when the non-blocking receive operation completes and the requested message is available in the application buffer.

```

MPI_Irecv (*buf, count, datatype, source, tag, comm, *request)
MPI_IRecv (buf, count, datatype, source, tag, comm, request, ierr)
    
```





Réseaux d'interconnexion dans les grappes de calcul

Brice Goglin
Brice.Goglin@ens-lyon.org

11 février 2005


Slides disponibles sur <http://perso.ens-lyon.fr/brice.goglin/teaching>

Cluster Design Issues

- Enhanced Performance (performance @ low cost)
- Enhanced Availability (failure management)
- Single System Image (look-and-feel of one system)
- Size Scalability (physical & application)
- Fast Communication (networks & protocols)**
- Load Balancing (CPU, Net, Memory, Disk)
- Security and Encryption (clusters of clusters)
- Distributed Environment (Social issues)
- Manageability (admin. And control)
- Programmability (simple API if required)
- Applicability (cluster-aware and non-aware app.)


Olivier Glück - © 2011



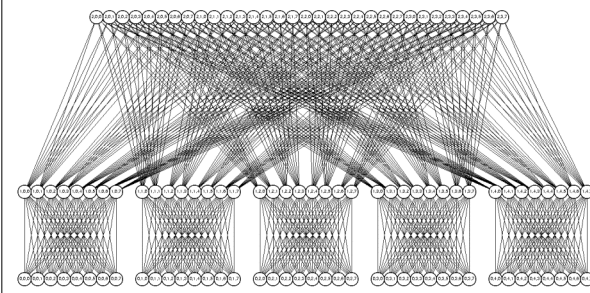
Topologie du réseau

- Réseau régulier
- Cœur suffisamment dimensionné
 - Tous les nœuds peuvent utiliser simultanément la capacité de leur lien
- Réseau de Clos
 - Full bisection bandwidth
- Routage statique
 - Les routes sont fixées à l'initialisation


Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 93



Réseau de Clos




Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 94



Routage par la source

- Routes fixées à l'avance
 - Stockées dans les cartes d'interface
- Route placée en tête des paquets
 - Rapide à traiter dans les switches
 - Intelligence laissée dans les cartes

Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 95

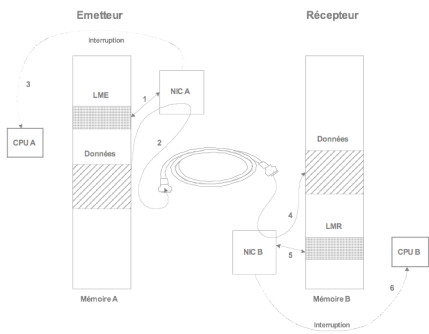


Fiabilité et traitement des erreurs

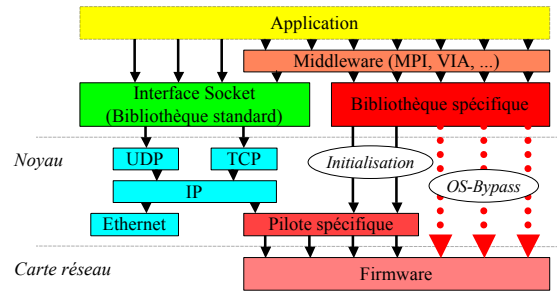
- Très peu d'erreurs
 - Peu de congestion
 - Cœur suffisamment dimensionné
 - Back-Pressure
 - Matériel fiable
 - Moins de 10^{-15} erreurs sur fibre optique
- CRC traités dans les cartes (voire les switches)
- Reprise sur erreur traitée par la carte

Olivier Glück - © 2011 M2 ENS - spécialité IF - Réseaux Avancés 96

La primitive d'écriture distante sur MPC



Modèle de fonctionnement



API traditionnelle : les Sockets

- Connexions entre nœuds
- Primitives bloquantes
 - read/write(fd, buf, size)
- Appels système
- Copie intermédiaire dans la mémoire système
 - Réduire le temps de blocage à l'émission
 - Gérer les reprises sur erreur
 - Gérer les données inattendues en réception

Clefs de la performance des grappes

- Utilisation maximale des différents nœuds et de leurs processeurs
- Communication rapide entre les nœuds
 - Faible latence
 - Grande bande passante
- Ne pas gaspiller les puissances des processeurs pour traiter les communications
- Recouvrir le traitement des communications par du calcul

Les techniques d'optimisation

- Eviter la traversée d'un grand nombre de couches de communication
- Réduire le nombre de copies des données
- Réduire le nombre d'appel système en réalisant les communications en espace utilisateur
- Eviter l'utilisation d'interruptions pour la signalisation des communications
- Réduire le coût des opérations de traduction d'adresses virtuelles/physiques

Consommation processeur

- Ne pas monopoliser les processeurs de calcul pour traiter les communications
 - Eviter les copies
 - DMA (*Direct Memory Access*) entre la carte et l'hôte
 - Réduire le coût du protocole
 - Utiliser un processeur dédié au réseau dans la carte d'interface

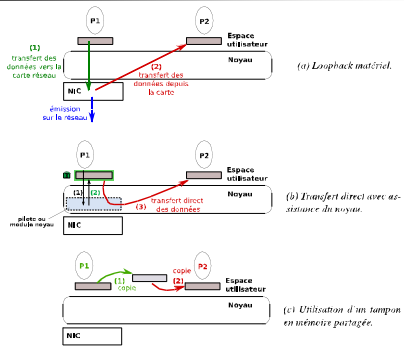
Recouvrement des communications

- Les communications prennent du temps
 - Ne pas bloquer l'application
- Traiter les communications pendant que l'application continue le calcul
 - Traiter les communications dans la carte d'interface
 - Utiliser des primitives non-bloquantes

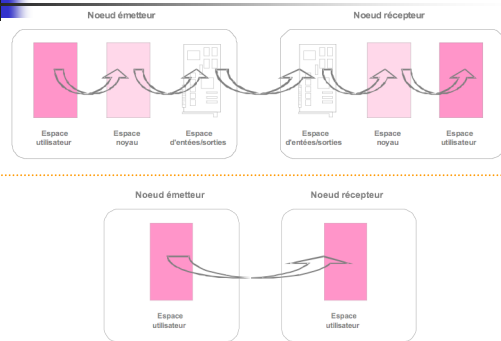
Déport de fonctionnalités dans la carte d'interface

- Protocole
 - Encapsulation
 - Scatter/Gather
 - Reprise sur erreur
- Transfert de données
 - Traduction d'adresses virtuelles en adresses physiques
 - DMA initiés par la carte

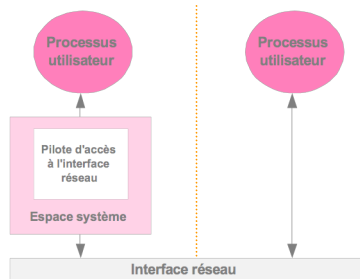
Techniques de communications entre noeuds



Une stratégie « zéro-copie »



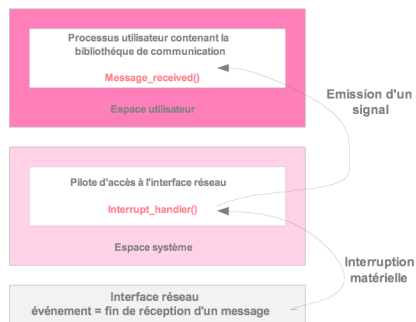
Les appels système



OS-Bypass

- Eviter de traverser toutes les couches système
 - Réduction du chemin critique
 - Appel système coûteux
 - Réduction du protocole
 - Protocole implémenté dans la bibliothèque et la carte
- Pas d'intervention du système d'exploitation
 - Aucune aide pour manipuler les adresses mémoire
 - Pas de synchronisation naturelle

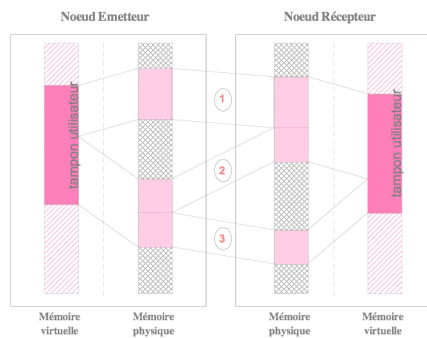
Les interruptions



Scrutation et interruption

- Interruption couteuse
 - De l'ordre de 10 μ s
 - Inutilisable pour obtenir une faible latence
- Scrutation de la carte
 - Attente active d'événements dans la carte
 - *Programmed I/O*
 - Utilisation d'interruptions par la suite
 - Après un certain temps de scrutation

Les traductions d'adresses



Enregistrement mémoire

- L'application passe des zones mémoires à la carte
 - L'application manipule des adresses virtuelles
 - La carte manipule des adresses physiques
 - Pour les DMA
- Nécessité de traduire sans l'aide du système d'exploitation
 - Enregistrement à l'avance des traductions dans la carte
 - Duplication de la MMU de l'hôte dans la carte

Memory registration/deregistration

Memory registration:

- Trap into OS kernel.
- Lock access to OS page table.
- Loop for every page:
 - Walk the OS page table to find the virtual page.
 - Eventually swap the virtual page in.
 - Increment the page reference count.
 - Pin virtual page in the OS page table (marked not swappable).
 - Get the IO address of the related physical memory page (may require to use the IOMMU).
- Unlock access to OS page table.
- Return to user space.

Memory deregistration:

- Trap into OS kernel.
- Lock access to OS page table.
- Loop for every page:
 - Walk the OS page table to find the virtual page.
 - Unpin virtual page in the OS page table (marked swappable).
 - Decrement the page reference count.
 - Eventually clear the related entry in the IOMMU.
 - Eventually clear the related entry in any cache on the NIC.
- Unlock access to OS page table.
- Return to user space.

Memory registration/deregistration

- Memory registration can be very expensive.
 - Hardware folks did not think that it would be used in the critical path: explicit memory registration in low-level hardware-driven hardware API such as VIA and Infiniband.
 - No explicit memory registration in higher level communication libraries such as MPI or Sockets.
- Various methods to attempt to dodge the bullet:
 - Not do zero-copy: make sense for small messages where memory copy is cheaper.
 - trashes cache, uses CPU.
 - Implement a registration cache: lazy deregistration with garbage collector.
 - Need to hijack malloc to catch when memory pages are released to the OS.
 - Maintenance nightmare.
 - Poor efficiency on complex code: 9% cache hit on Linpack.
 - Do not register memory: maintain a copy of the OS page table in the NIC.
 - No OS support: requires patching the OS, portability issues.

Message Passing et RDMA

- MPI basé sur *Rendez-vous*
 - Notification des deux côtés
- *Remote Direct Memory Access*
 - Accès à la mémoire d'un autre nœud
 - Notification uniquement pour l'initiateur
 - Le nœud distant n'est pas informé
- Difficile d'implanter MPI sur RDMA
 - Passage à l'échelle très difficile : scrutation sur les tampons de réception

Message Passing vs. RDMA

- Hardware folks like simple semantics like PUT or GET.
 - Called RDMA by the marketing department.
- Software guys use two-sided interfaces such as MPI or Socket.
 - Two-sided interfaces are easier to manipulate for large, complex code.
 - MPI is the de facto programming standard in HPC.
- Mapping MPI on top of RDMA is like:
 - Train an AI to be a shrink.
 - Using HPF to generate non-trivial parallel code.
 - Running HPC codes on a set of loosely coupled, geographically widely distributed machines.

It sounds easy but it is a pain to implement and it performs poorly.

Message Passing over RDMA

- History repeats itself: Memory Channel, Gigaset (VIA), SCI, IB.
- How to implement a matching semantic on top of a one-sided API?
 - Matching as to be done somewhere sometime by someone.
- Matching can be done after sending data:
 - Eager mode, copy on the receive side.
 - Where do you PUT the eager message in the first place?
 - Shared queue with tokens? Multiple windows? Polling or blocking?
- Matching can be done before sending data:
 - Rendez-vous with small packets containing matching information.
 - Matching done by the host whenever the rendez-vous happen.
 - If the host is not available, either wait or interrupt it.

Exemples de réseaux rapides de grappes



Prominent Components of Cluster Computers

- High Performance Networks/Switches
 - Ethernet (10Mbps),
 - Fast Ethernet (100Mbps),
 - Gigabit Ethernet (1Gbps)
 - ATM (Asynchronous Transfer Mode)
 - SCI (Scalable Coherent Interface)
 - Myrinet
 - QsNet (Quadrics Supercomputing World)
 - Digital Memory Channel
 - FDDI (fiber distributed data interface)
 - InfiniBand

Prominent Components of Cluster Computers

- Fast Communication Protocols and Services (User Level Communication):
 - Active Messages (Berkeley)
 - Fast Messages (Illinois)
 - U-net (Cornell)
 - XTP (Virginia)
 - Virtual Interface Architecture (VIA)
 - SISCi (SCI), GM et MX (Myrinet), Verbs (Infiniband), ElanLib (QsNet), ...

Scalable Coherent Interface (SCI)

- Créé par Dolphins en 1992
- Pont entre l'hôte et le réseau pour créer une mémoire globale
- Topologie en anneaux ou tores 3D
 - Partage de bande passante
 - Peu scalable
- Communication par lecture ou écriture en mémoire physique distante
- Au niveau Software Interface for SCI :
 - RDMA
 - Mémoire partagée (projection dans la mémoire locale du processus des segments de mémoire distants)

Scalable Coherent Interface (SCI)

- Latence et débit au niveau SISI
 - 340 Mo/s
 - Latence RDMA : 1.4 μ s
 - Latence mémoire partagée : 3.8 μ s
- Pas de processeur dédié sur la carte d'interface
 - Grande utilisation du processeur de l'hôte
- 0-copie, mais pas OS-bypass
- Verrouillage des pages en mémoire
- Table des correspondances virtuelles/physiques dans la carte
- Possibilité de déclencher une interruption sur le noeud distant

Myrinet

- Myricom, leader du marché, existe depuis 1994
- Conçu pour le *Message Passing*
- Cartes et switchs facilement programmables
 - Processeur RISC à 333 MHz (LANai) + 2 Mo de SRAM + moteur DMA embarqués sur la carte
 - Beaucoup d'interfaces logicielles différentes
- Utilise surtout Linux
- 2*250 Mo/s et 2,5 μ s
- Environ \$1000 par noeud

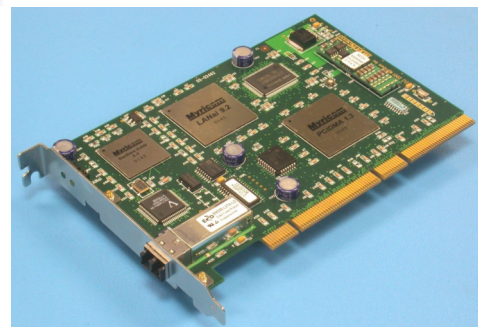
Myrinet

- Topologie en clos basé sur gros switchs
 - *Full bisection bandwidth*
 - Clusters de plus de 2000 noeuds (MareNostrum - 2282 noeuds au Barcelone Supercomputing Center)
- Routage dispersif par la source
 - Plusieurs routes par destination
 - Répartition du trafic pour homogénéiser la charge dans le réseau
- Vérification des erreurs et contrôle de flux réalisés par le matériel
- Port Ethernet sur les derniers switch

Myrinet

- Enregistrement mémoire pour 0-copie
 - Pages verrouillées
 - Cache de traduction d'adresse dans la carte
 - Initialement exposé à l'application
 - Beaucoup trop d'inconvénient
 - Désormais utilisé en interne si nécessaire
- *Message Passing* et RDMA logiciels dans la NIC
 - Adapté aux applications parallèles
- Une nouvelle interface logicielle (MX) très proche de MPI + Socket MX

Myrinet



Myrinet

Switch 256 ports extensible



Quadrics QsNet

- Matériel très performant et très cher
 - 900 Mo/s et 2 μ s
 - Environ \$2000 par nœud
- Topologie en clos basé sur des petits switches (jusqu'à 64 ports)
- Fonctionnalités réseau dans le matériel
 - Broadcast/Multicast
- Spécifications du matériel non ouvertes

Quadrics QsNet

- MMU dupliquée sur la carte
 - Modification du système d'exploitation pour la maintenir à jour
 - 0-copie, OS-bypass sans enregistrement mémoire
- Interface bas niveau basée sur RDMA (ElaLib)
 - avec notification distante
- Interface de *Message Passing* au dessus (TSPORTS)

Infiniband

- Norme récente définie par de nombreux constructeurs
- Devait initialement fournir un standard de bus I/O, réseau, stockage, ...
- Peu à peu délaissé par les grands constructeurs
- Cible désormais uniquement le réseau hautes performances
 - Communications et stockage

Infiniband

- Protocole bas-niveau proche de IP
 - Interopérable
 - Routage dans les switches
- Bande passante énorme annoncée
 - 4x = 1 Go/s, 12x arrive, 30x annoncé
- Latence 5 μ s
- Entièrement basé sur RDMA
 - Pas conçu pour les applications de type MPI (pas de notification sur le nœud distant)
- VERBS : interface logicielle proche de VIA

Et Ethernet dans tout ça ?

- 10G Ethernet arrive
- Approche 10 μ s de latence
- Déport de TCP/IP dans la carte
- Utilisation du processeur de l'hôte reste trop grande
- Difficile de faire des gros switches

Récapitulatif

Interconnect	Débit (Mo/s)	Latence (μ s)	Coût	Topologie	Support MPI
Ethernet	100+100	50	Faible	Variable	Oui
Quadrics	900+900	2	Très élevé	Clos	Oui (4.6 μ s, 308MB/s)
Infiniband	800+800	5	Élevé	Clos ou autre	Oui (6.8 μ s, 841MB/s)
SCI	340 partagés	2	Moyen	Anneau ou Tore	Oui
Myrinet/GM	500+500	3	Moyen	Clos	Oui (6.7 μ s, 235MB/s)

Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

133

Récapitulatif

Interconnect	API bas-niveau	Interruptions	Recouvrement calcul/communication	Utilisation CPU
Ethernet	Socket	Oui	Oui	Grande (copie)
Quadrics	RDMA + Notification	Non	Partiel	Faible (DMA) Overhead MPI 3.3 μ s
Infiniband	RDMA	Non/Oui	Non	Faible (DMA) Overhead MPI 1.7 μ s
SCI	RDMA	Oui	Non	Grande (copie PIO)
Myrinet/GM	Message Passing	Non	Non	Faible (DMA) Overhead MPI 0.8 μ s

Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

134

Récapitulatif

Interconnect	Zéro-copie	Enregistrement mémoire	Cache d'enregistrement	Patch de l'OS
Ethernet	Non	Non	Non	Non
Quadrics	Oui sauf petits messages	Non	Non	Oui
Infiniband	Oui sauf petits messages	Oui	Oui	Non ?
SCI	Non	Oui	Oui	Non
Myrinet/GM	Oui sauf petits messages	Oui	Oui	Non

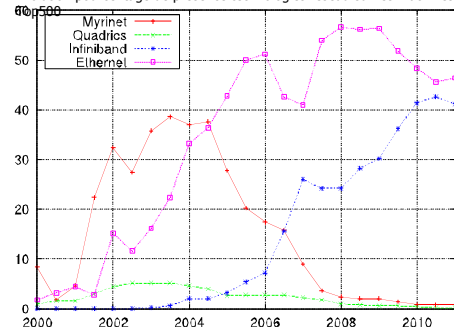
Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

135

Evolutions réseaux rapides

Evolution pourcentage de présence technologies réseau sur les machines du

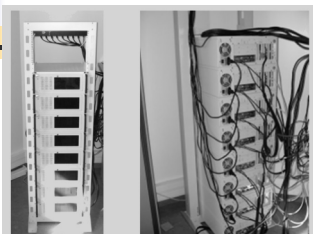


Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

136

Optimisations de la bibliothèque de communication MPI pour machines parallèles de type « grappe de PCs » sur une primitive d'écriture distante



Olivier Glück
UPMC/LIP6/ASIM

CS
CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

INRIA

Olivier.Gluck@lip6.fr



Contexte et hypothèses

- Machines parallèles de type grappe de PCs sous UNIX
- Applications MPI en mode « batch »
- Réseau de communication : primitive d'écriture distante
 - le réseau est fiable
 - le contrôleur réseau utilise des accès DMA en lecture et en écriture pour accéder à la mémoire du nœud hôte
 - le contrôleur réseau ne peut transférer que des zones contiguës en mémoire physique
 - le nœud émetteur doit savoir à l'avance où déposer les données sur le nœud récepteur

Olivier Glück - © 2011

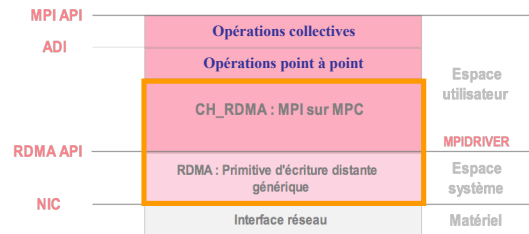
M2 ENS - spécialité IF - Réseaux Avancés

138

Objectifs

- Fournir l'environnement MPI :
 - machines parallèles de type « grappe de PCs »
 - primitive d'écriture distante
- Réduire le chemin critique logiciel

MPICH : une architecture en couches



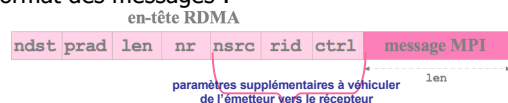
L'API RDMA

RDMA : *Remote Direct Memory Access*
API générique d'écriture en mémoire distante

- Objectifs :
 - définir la brique de base de notre implémentation de MPI
 - masquer les particularités des réseaux fournissant une primitive d'écriture distante
- Enjeu :
 - faire en sorte que notre travail puisse être exploitable sur d'autres plate-formes matérielles

L'API RDMA

- L'écriture distante :
 - RDMA_SEND(nsrc, ndst, plad, prad, len, ctrl, sid, rid, ns, nr)
- La notification :
 - RDMA_SENT_NOTIFY(ctrl, sid)
 - RDMA_RECV_NOTIFY(nsrc, ctrl, rid)
- La signalisation par scrutation (optionnelle) :
 - RDMA_NET_LOOKUP(blocking)
- Format des messages :

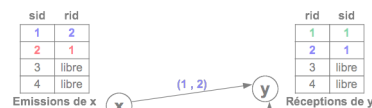


Les problèmes à résoudre

- Les problèmes liés à la primitive d'écriture distante :
 - le dépôt direct en mémoire : l'émetteur doit connaître les adresses physiques des tampons distants
 - l'utilisation d'adresses physiques alors que l'application manipule des adresses virtuelles
- Les services à fournir à MPICH :
 - transmission des messages CTRL/DATA
 - signalisation des événements réseau
 - mécanisme de contrôle de flux
 - fixer le seuil optimal entre les messages CTRL/DATA

Les émissions/réceptions simultanées

- Pourquoi : pouvoir traiter plusieurs émissions/réceptions simultanément -> besoin d'identifier une communication



On utilise des tables associatives :

- sid choisi par l'émetteur
- rid choisi par le récepteur

1 canal (sid,rid) à la durée de vie d'un message

sid	rid
1	1
2	libre
3	libre
4	libre

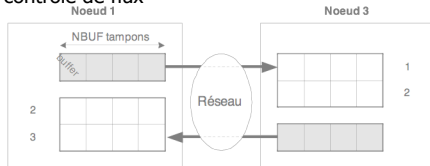
Emissions de z

rid	sid
1	2
2	libre
3	libre
4	libre

Réceptions de z

Les messages CTRL

- Transfert d'informations de contrôle ou de données de taille limitée
 - Utilisation de tampons intermédiaires contigus en mémoire physique et alloués au démarrage de l'application
- > pas de traduction d'adresses, adresse du tampon distant connue sur l'émetteur, recopie systématique, contrôle de flux



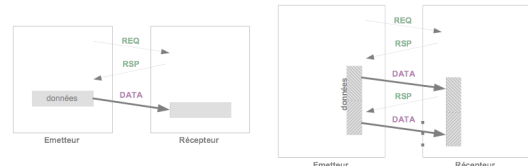
Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

145

Les messages DATA

- Transfert des données de l'application en mode « zéro-copie »
 - Utilisation d'un protocole de rendez-vous entre l'émetteur et le récepteur
- > traduction d'adresses sur l'émetteur et le récepteur, le message RSP contient la description en mémoire physique du tampon de réception



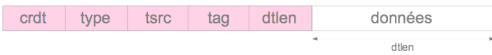
Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

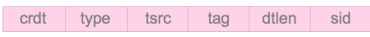
146

Le format des messages CTRL

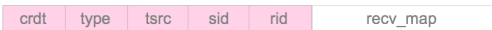
SHORT : 20 octets + dtlen



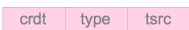
REQ : 24 octets



RSP : 20 octets + 8*NB_DMA



CREDIT : 12 octets

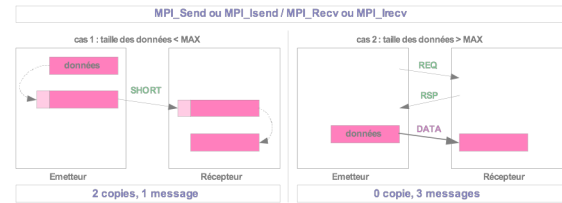


Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

147

Le mode standard dans MPI

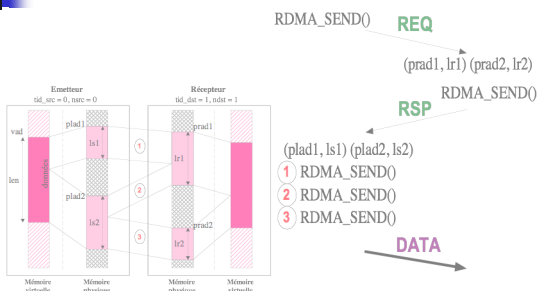


Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

148

Liens entre MPI et l'API RDMA



Olivier Glück - © 2011

M2 ENS - spécialité IF - Réseaux Avancés

149