3ème année PRCD & RSR

IT 352

Systèmes parallèles et distribués

Brice Goglin

2011-2012

Copyright

Ceci n'est PAS un polycopié de cours.

Ces « notes » guident simplement le cours.

De nombreuses explications et illustrations manquent.

Les détails seront données au tableau et à l'oral pendant le cours magistral.

Programme du module

- 5 séances de cours
- 2h40 de TP, en salle machine
  - DSM en espace utilisateur
- Slides mis à jour en ligne au fur et à mesure
  - http://runtime.bordeaux.inria.fr/goglin/teaching/SPD.html
- Examen écrit de 2h
  - Tous documents autorisés

Plan du cours

- Généralités et concepts
- Mémoire partagée distribuée
- Systèmes d'exploitation à image unique
- ~~Stockage distribué~~

Pour me joindre

- Equipe-Projet INRIA Runtime
  - A29bis, préfabriqués INRIA en face du LaBRI
- 05.24.57.40.91
- Brice.Goglin AT inria.fr
- http://runtime.bordeaux.inria.fr/goglin/

Introduction

Introduction historique

- Après la guerre
  - Apparition des premiers ordinateurs
  - Gros calculateurs centralisés, très coûteux
- Début des années 80
  - Micro-ordinateurs produits en grande quantité, peu chers
  - Faciles à interconnecter
    - 1 ou 10Mb/s, Ethernet ou Token Ring
  - Quels logiciels pour les utiliser ?

Intérêt des systèmes distribués

- Mise en commun d'un grand nombre de ressources à faible coûts
  - Bon rapport performance/prix
  - Puissance globale virtuellement illimitée
    - Supérieure à celle des gros calculateurs
- Disponibilité et flexibilité
  - Un élément peut tomber en panne sans bloquer tout le système
  - Distribution de la charge

Intérêt (2)

- Réponse à la distribution géographique de certains systèmes existants
  - Réplication locale des ressources globales
- Partage de ressources coûteuses entre plusieurs utilisateurs/machines
  - Accès à distance à une ressource indisponible en local
  - Accès aux mêmes ressources depuis tous les endroits du système

Inconvénients des systèmes distribués

- Logiciels de gestion difficiles à concevoir
  - Peu d'expérience ou succès dans ce domaine
  - Complexité imposée par la transparence
- Problèmes inhérents aux communications
  - Communications explicites si pas de mémoire partagée
  - Lenteur, saturation, perte de messages
- Partage et distribution de données impose mécanismes complexes
  - Synchronisation
  - Sécurité

Le plus grand système distribué actuel

- Internet
- Contient de nombreux sous-systèmes selon le protocole considéré
  - Web (http)
  - Bittorrent (peer-to-peer)

Super-calculateur ?

- Seti@home !
  - Les machines de tout le monde
    - Puissance virtuellement illimitée
      - N Gflop/s par machine, des millions des machines
  - Limité à des problèmes très simple
    - Calculs indépendants
    - Pas de communication ou synchronisation entre les instances
    - Modèle plus distribué que parallèle

Différences entre systèmes parallèles et distribués

- Architecture matérielle et/ou logicielle
  - Un système parallèle peut être distribué
    - Si assemblage de machines à mémoires indépendantes
  - ou non
    - Si une seule grosse machine parallèle (virtuelle?)
- Application cible
  - Applications parallèles dans les systèmes parallèles
    - Communication, synchronisation, … entre les différentes instances

Super-calculateurs

- Cray Jaguar à ORNL
  - 20 000 noeuds à 2 opterons six-core
    - Assemblés par réseau Cray spécifique
      - Presqu'un Cluster, mais réseau trop spécifique
  - Plus d'1 PetaFlop/s

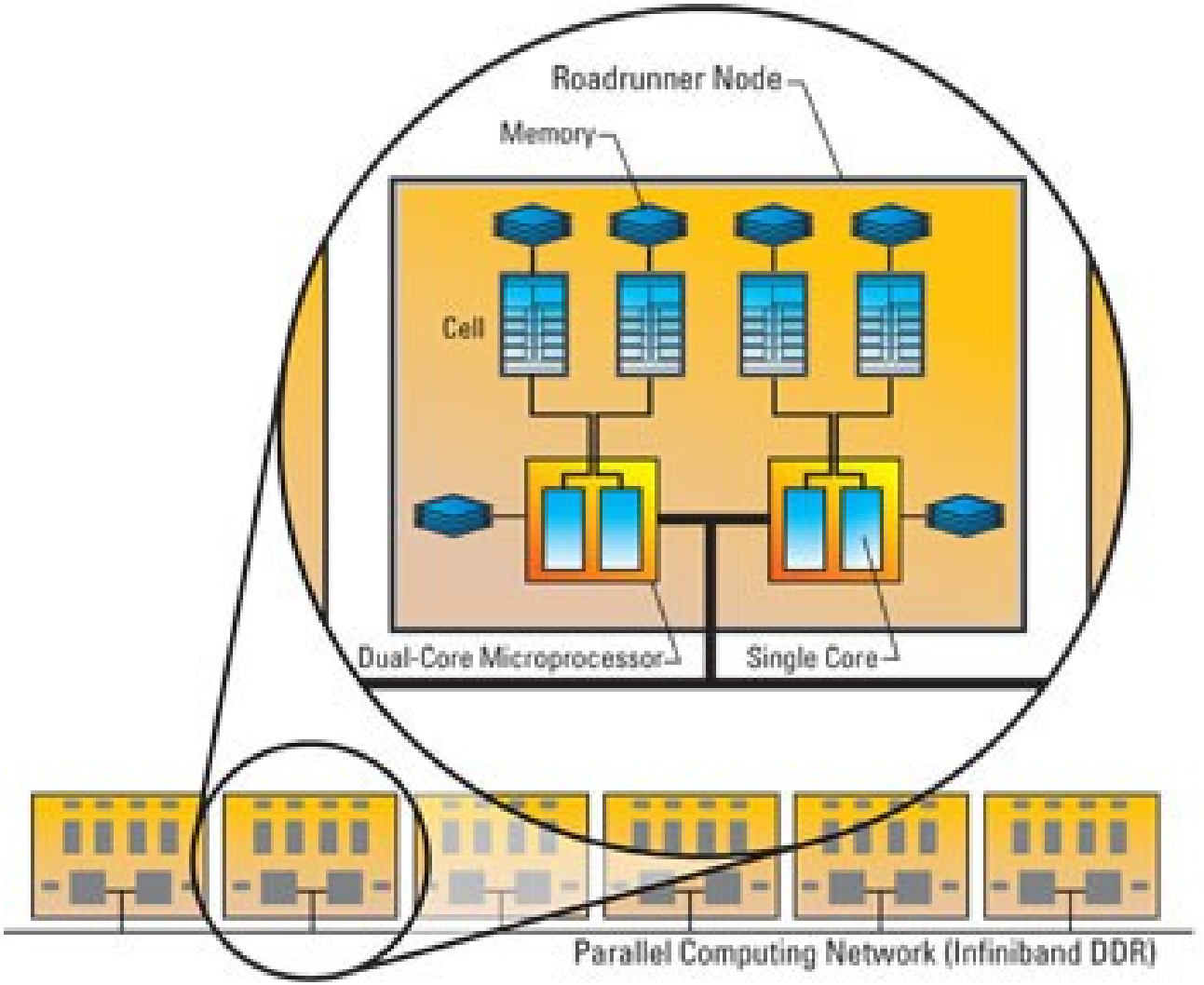
Super-calculateurs (2/4)

- Tianhe en Chine
  - 7000 nœuds à 2 processeurs et 2 GPUs
  - Un cluster avec des accélérateurs
  - 2 Petaflop/s

Super-calculateurs (3/4)

- RoadRunner à Los Alamos
  - Le premier supercalculateur > 1 PFlop/s
  - 3000 noeuds bi-Opterons
  - 12000 cartes d'extension avec des processeurs Cell (8 coeurs spécialisés)
    - Le cluster d'opteron est surtout un réseau d'interconnexion et gestion des Cells

Architecture de RoadRunner

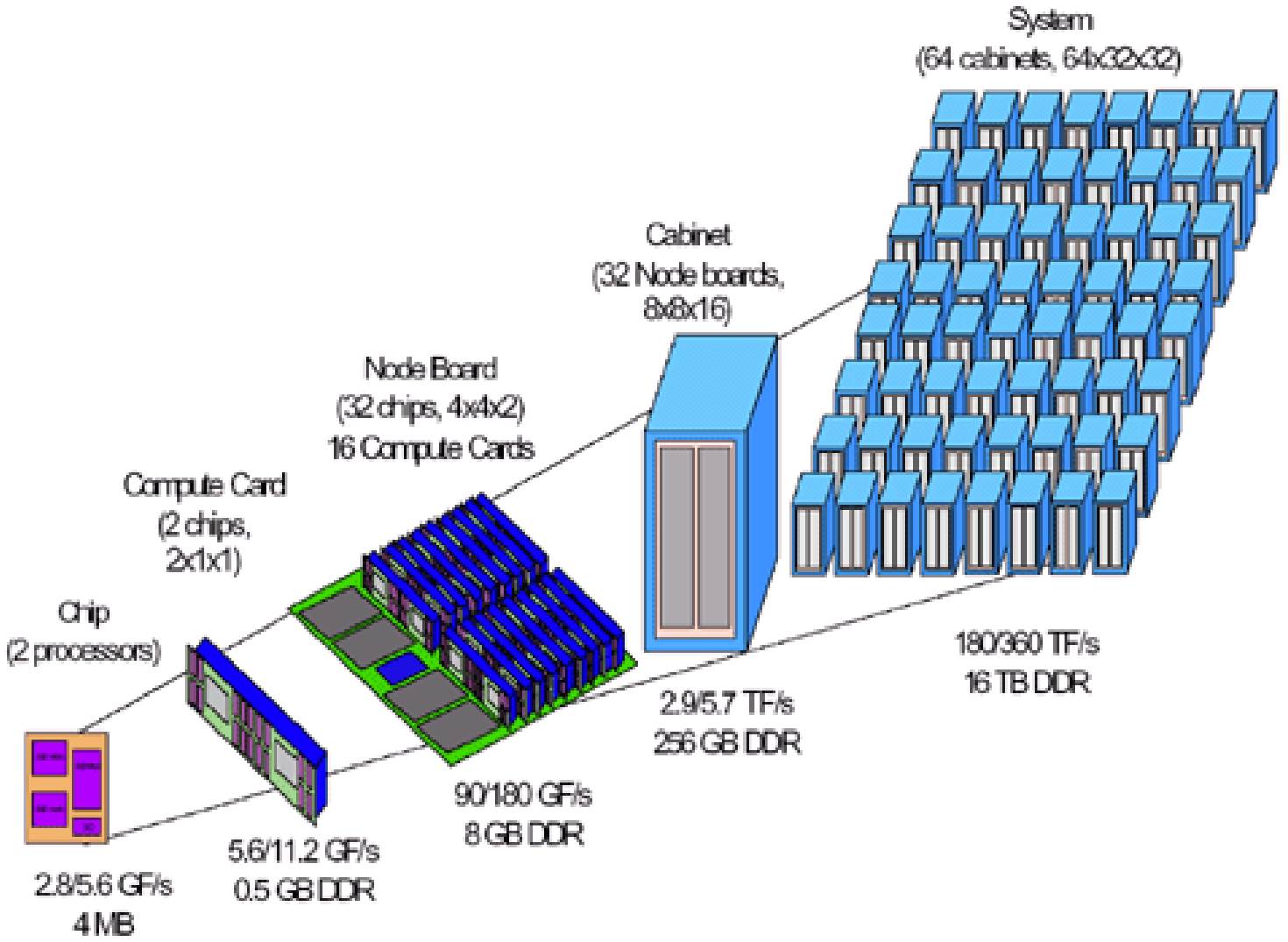


Super-calculateurs (4/4)

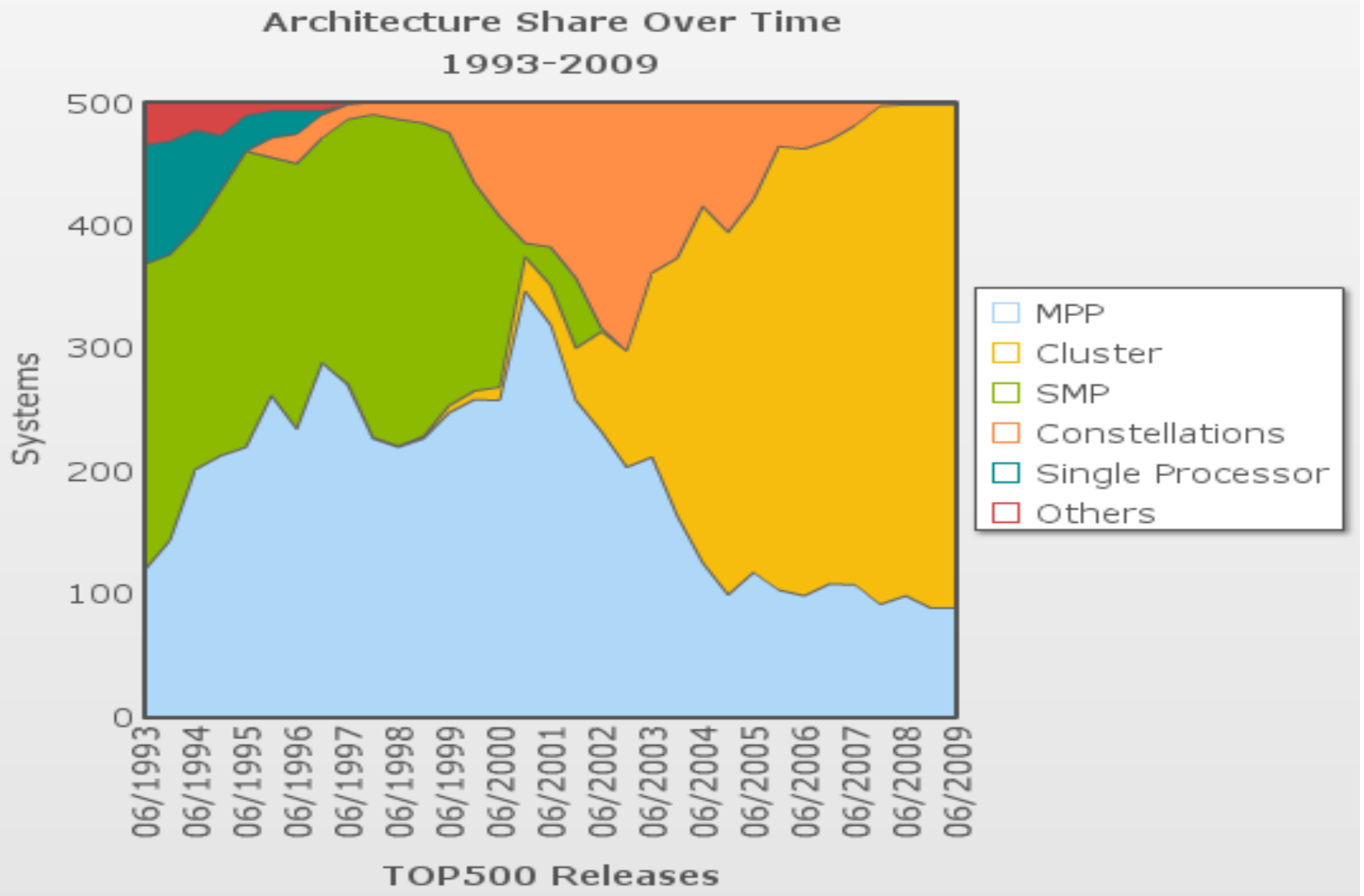
- BlueGene (IBM)
  - Enormément de *Blades* de calcul très simples
    - Centaines de milliers de coeurs
    - Processeurs et mémoire uniquement
    - Matériel peu fiable (disque) ou inutile (USB) supprimé
      - Moins de consommation, moins de pannes
  - 1 PFlop/s actuellement (BG/P)
    - En attendant la génération suivante

Architecture BlueGene

Architecture BlueGene (2/2)



Évolution des systèmes parallèles



Évolution des systèmes parallèles (2/2)

- Au début, des très gros calculateurs
  - Massivement parallèles (MPP)
  - Matériel dédié
    - Cher mais efficace
- Depuis 15 ans, grappes de calcul (Clusters)
  - Assemblage de machines
  - Matériel plus standard
    - Moins efficace mais beaucoup moins cher

Tendances actuelles

- Des systèmes de plus en plus complexes
  - NUMA, Multicoeurs, accélérateurs
    - Coeurs de calcul généralistes, GPU, Cell, ...
- Parallélisme interne vs. externe
  - Mémoire partagée ou non
    - Même à l'intérieur des noeuds

Problèmes

- Consommation d'énergie
  - Plusieurs megawatts pour les plus gros calculateurs
    - Green500.org vs. Top500.org
    - Le Cell et BlueGene sont très bons
- Des systèmes de plus en plus distribués
  - De plus en plus de machines assemblées
  - Tolérance aux pannes de plus en plus critique
    - On ne sait pas comment atteindre l'ExaFlop/s

Différents niveaux de complexité

- Complexité interne et cohérence matérielle
  - Machine mono-processeur, SMP, NUMA, ...
  - Concurrence, synchronisation, ...
  - Accès non-uniforme aux données, affinité, ...
- Complexité externe
  - LAN et grappes, WAN et grilles, ...
  - Communications coûteuses, accès indirect aux données, migration, ...
- Hétérogénéité

Comment gérer cette complexité ?

- Quelles responsabilités et charges de travail pour l'OS et pour l'utilisateur ?
  - Et pour le matériel ?
- La complexité doit-elle être complètement masquée ?
  - Accès mémoire distant implicite ?
  - Communication implicite ?
  - Hétérogénéité masquée ?

Généralités et concepts

Architecture

- Ensemble de processeurs interconnectés
  - Différents types d'interconnexion
    - Bus, hiérarchique, switché, ...
      - Bus = peu scalable
      - Hiérarchique = solution onéreuse dépassée
      - Switché = AMD HT, Intel QPI = le futur ?
      - Effets NUMA partout
    - Interne ou externe à la machine
    - Géré en hardware ou en software
  - Différentes façon de communiquer
    - Mémoire partagée ou privée

Mémoire distribuée

- Mémoire privée à chaque machine
  - Mémoire inaccessible directement à distance
  - Nécessite migration/copie de pages
    - Problème de cohérence et partage
  - Utilisation de réseaux traditionnels ou rapides
    - Ethernet, Myri-10G, Infiniband, Quadrics, ...
  - Systèmes faiblement couplés

Systèmes parallèles ou systèmes distribués ?

- Système distribué
  - Partage de ressources
  - Tâches indépendantes
    - Réseau de stations de travail
  - Modèle client-serveur
- Système parallèle
  - Mise en commun de ressources pour exécuter application gourmande en calcul et/ou mémoire
  - Mémoire partagée et/ou réseau de machines
- Frontière ?

Systèmes distribués à grande échelle

- Machines quelconques
  - Réseau local de stations de travail
    - voire géographiquement distribué
  - Grappes de calcul
- Reliées par réseau traditionnel
  - Longue distance, topologie complexe
  - Performances faibles et variables
  - Pare-feu, authentification, sécurité, ...

Communications dans les systèmes distribués

- Beaucoup de besoins différents
- Beaucoup d'architectures matérielles différentes
- Beaucoup de fonctionnalités logicielles différentes
- Beaucoup de types de communication différents
  - Protocoles de haut niveau
  - Couches de bas niveau logiciel et/ou matériel

Communication entre instances d'un système distribué

- Systèmes à image unique
  - Matériel ou logiciel
- Mémoire partagée
  - Mécanismes de synchronisation
  - Matériel ou logiciel
- Échange de messages
  - Protocole prédéfini dans l'application
  - Modèle client-serveur
- Appel de procédure à distance
- Bloquant ou non, synchrone ou non

Communications explicites *Message Passing Interface*

- MPI règne dans les systèmes parallèles
  - Passage de messages
  - Rendez-vous
  - Matching pour filtrer
  - Primitives bloquantes ou non
  - Opérations point-à-point et collectives

Communications explicites *Remote Direct Memory Access*

- RDMA disponible dans matériel HPC (InfiniBand)
  - Utilisé en dessous de MPI selon le matériel
- Cible ouvre des fenêtres RDMA et donne identifiant aux autres
  - Les autres peuvent lire/écrire dedans
- Modèle « One-sided »
  - Seul l'initiateur de la communication sait qu'elle a lieu et quand elle se termine
    - Aucune synchronisation
      - Bien si synchronisation ad-hoc par autre moyen

Systèmes d'exploitation distribués

- Interface entre matériel et utilisateur
- Lien étroit avec le matériel
  - La conception du système dépend de l'architecture
    - Faiblement ou fortement couplé
  - Le système d'exploitation définit la vue de l'architecture par l'utilisateur
    - Ressources virtualisées accessibles de manière transparente

Systèmes d'exploitation distribués (2)

- Lien étroit avec la façon de programmer
  - Conception du système dépend des applications cibles
    - Application parallèles/couplées ou indépendantes
      - Threads, mémoire partagée, MPI, quelconque, ...
  - Applications programmées selon fonctionnalités du système d'exploitation
    - Ressources partagées et accessibles de manière transparente
      - Threads + mémoire partagée ont fortes contraintes
      - MPI a des contraintes faibles

Systèmes d'exploitation pour machine multi-processeur

- Une seule copie de l'OS sur la machine
  - Synchronisation pour l'accès aux structures partagées du noyau
- Exécution simultanée de plusieurs tâches
  - Chaque tâche se croit isolée
- Support immédiat de
  - Mémoire partagée
  - Migration des tâches

Système distribué collaboratif

- OS faiblement couplé
  - Machines indépendantes
  - Exécution de tâches possible sans le réseau
  - Réseau permet partage de ressources
- Un OS sur chaque machine
  - OS standard + services distribués
  - Modèle client-serveur
  - Hétérogénéité
- NOW, grilles, Internet, ...

Distribution matérielle ou logicielle ?

- Matériel distribué
  - Mémoire et processeurs pas directement accessibles
    - Réseau de communication sans accès direct à la mémoire
  - Mémoire et processeurs rendus accessibles par matériel spécialisé
    - Sans intervention du logiciel

Distribution matérielle ou logicielle ? (2/2)

- Compensation possible par le logiciel
  - Donner l'illusion d'un seul système au dessus de matériel distribué
    - Communications bas-niveau explicites effectuées de manière transparente
      - Par OS ou bibliothèque

Système à image unique

- Vision d'une unique machine virtuelle
  - Regroupe toutes les ressources de machines physiques
- Transparent pour l'utilisateur
- Nœuds anynomes
- Haute disponibilité
- Pas forcément centralisé
- Impose synchronisation forte entre les différentes instances du système

SSI matériel

- Agrégation matérielle de machines
- Partage transparent des processeurs et de la mémoire
  - Illusion d'une grosse machine NUMA
- SGI NUMAlink

SGI NUMAlink

Photo NASA

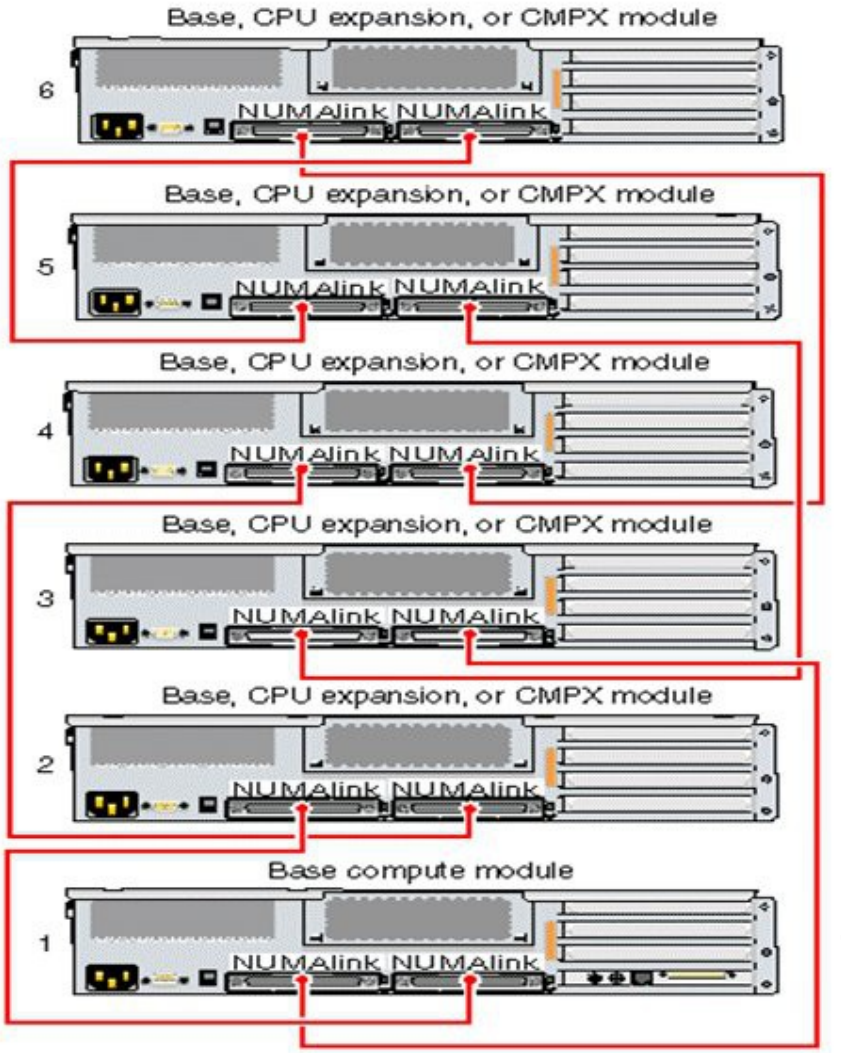


Schéma SGI

SSI matériel (2/2)

- Intégralement géré par le matériel
  - Accès transparent à mémoire distante
    - Déréférencement de pointeurs vers mémoire physique distante
  - Cohérence de cache
- Pas de migration des données
  - Sauf pour le cache
    - Gros travail sur le maintien efficace de la cohérence entre caches
      - Eviter les broadcasts sur les très grandes machines
      - « Directory » pour savoir quel cache contient quelle ligne

Support dans le matériel pour mémoire partagée

- Cartes réseau permettant le *Mapping* d'adresses physiques distantes dans l'espace virtuel local
  - Dolphin SCI
  - Possibilité de déréférencer pointeur distant
    - DSM transparente pour l'utilisateur
- Nécessite de mettre en place le mapping
  - Dans le pilote (dans l'OS)
- Espace mappable limité

Cohérence de cache

- La mémoire peut être accessible directement (déréférençable)
  - Mais sans cohérence de cache
    - Même à l'intérieur d'une seule machine
      - Heureusement, c'est rare !
        - Mais ça pourrait revenir à la mode à moyen terme
- Synchronisation des caches nécessaire pendant accès concurrents

Cohérence dans SCI

- Peu d'opérations atomiques distantes
  - atomic_add_and_fetch
- Ordre des transactions pas conservé
- Mémoire distante non-cachable
  - Lent ou non-cohérent

- Travail nécessaire pour la mise en œuvre d'une DSM cohérente

Coprocesseurs de calcul

- GPGPU, Cell BE, Clearspeed, FPGA, ...
- Périphérique traditionnel (PCI-e)
- Très grosse puissance de calcul
  - Jusqu'à 1Tflop/s par carte
- Mais très basique aussi
  - Incapable de faire des travaux complexes
  - Uniquement du calcul de base
    - Déjà très utile !

Coprocesseurs de calcul (2/2)

- Mémoire embarquée
  - Indépendante de la mémoire principale
    - Inaccessible directement (généralement pas déréférençable)
- Nécessité de déplacer les données entre la machine et le coprocesseur (DMA ?)
  - Migration explicite de données
  - Cohérence, synchronisation
- Comme un système distribué !
  - Mais à l'intérieur d'une même machine

Beaucoup de niveaux de complexité

- Mono-processeur
- SMP
- CC-NUMA
- NCC-NUMA
- Coprocesseur avec mémoire embarquée

- Distribué masqué par le matériel
- Distribué masqué par le logiciel
- Distribué

Mémoire partagée distribuée

Concepts

- DSM (*Distributed Shared Memory*)
- Partage de mémoire entre machines
  - Au niveau du système ou des applications
- Modèle de programmation simple
  - Surtout pour les applications parallèles
    - Bien pour threads, bof si communication explicite
- Exploitation de la mémoire distante
  - Pagination à distance
  - Cache collaboratif

Rappels sur la pagination

- Espace d'adressage virtuel des processus divisé en pages de taille fixe
  - Une table des pages par processus
- Page virtuelle associée à
  - Pages physiques dans la mémoire principale
  - Blocs disques dans un espace de *Swap*
- Traduction virtuel→physique transparente
  - MMU du processeur
  - Défauts de page

Algorithmes de pagination

- Si mémoire saturée, évincer des pages sur le disque (*Swap-out*)
- Choisir les bonnes pages
  - LRU
  - FIFO
  - ...
- Possibilité de pré-évincer pour améliorer les performances (*Pré-Swapping*)
  - Regrouper les accès aux disques

Algorithmes de pagination (2)

- Si la page est sur le disque, la ramener en mémoire principale (*Swap-in*)
- Évincer une autre page sur le disque
- Possibilité des précharger pour diminuer le délai d'attente (*Pre-faulting*)
  - Regrouper les accès aux disques

Pagination en mémoire distante

- *Swap* à distance
  - Plus rapide que sur disque
    - Dépend beaucoup réseau
      - réseau rapide (10us + 1Go/s) < disque (5ms + 100 Mo/s)
  - Moins fiable
- Utile pour applications irrégulières
  - Peu pour MPI
    - Utilise souvent toute la mémoire sans swapper
      - de manière homogène dans le temps et l'espace
    - Calcul *out-of-core*

Global Memory Service (GMS)

- Utiliser la mémoire des autres nœuds comme cache entre la mémoire locale et le disque
- Gestion de l'ajout/retrait de nœuds
- Migration des pages non-critiques
  - Pages privées du processus
  - Pages de fichiers partagés
  - Les pages utilisées par l'OS ne migrent pas
    - Pas de risque pour l'OS local

Global Memory Service (2)

- Politique LRU globale
- Ajustement dynamique des quantités de mémoire globale et locale
- Nœuds actifs
  - Surtout de la mémoire locale
    - Pages moins récentes évincées vers autres nœuds
- Nœuds inactifs
  - Surtout de la mémoire globale
    - Pages anciennes évincées vers disque
    - Stockage des pages des autres nœuds

Global Memory Service (3)

- Identifier pages anciennes sans centraliser un état global
- Informations approximatives sur pages globales
  - Informations précises à chaque début d'époque
  - Changement d'époque après un délai ou un nombre de migration maximum
- Le nœud actuellement le moins actif prend en charge la politique

Global Memory Service (4)

- GMS mis en œuvre dans OSF-1
- Mémoire virtuelle des processus et cache de fichiers unifiés
- Noyau modifié pour tenir compte de GMS lors de l'ajout/retrait de pages
- Gestionnaire de cache global
  - Localiser une page
- Gestionnaires de cache locaux
  - Manipuler une page

Mémoire partagée distribuée

- Mémoire partagée mise en œuvre sur mémoire distribuée matérielle
- Soit par le système d'exploitation ou les *middleware*
  - Utilisable par toutes les applications
  - Pas optimale
    - Nécessite indications sur besoins de l'application
- Soit par l'application elle-même
  - Adapté aux besoins de cette application
  - Pas réutilisable

Retour sur GMS

- Mémoire matériellement distribuée et logiciellement partagée
- Uniquement visible pour les noyaux
  - L'application voit plus de mémoire physique mais ne peut pas facilement l'exploiter
- Pas conçu pour partage mémoire entre applications
  - Conçu pour pagination (*Swap*) à distance

Objectifs des DSM

- Agréger mémoire matériellement distribuée
- La rendre déréférençable par tout le monde
- Cohérence de cache
  - Partage de données entre applications
  - Opérations atomiques

Déréférencer de la mémoire distante ?

- Impossible dans le cas général !
  - Un processeur ne sait déréférencer que la mémoire locale
    - Et certains bouts de la mémoire des périphériques
- Possible avec aide du matériel
  - SGI Numalink
    - Avec un SSI matériel, c'est de la triche
  - Mapping distant SCI
    - Mais très limité en espace
    - Pas de cohérence de cache
      - Travail logiciel pour assurer la cohérence

Contraintes des DSM logicielles

- Matériel ne peut pas accéder directement à mémoire distante
  - Pas de déréférencement explicite
  - Nécessité de migrer les données localement avant l'accès par le processeur
    - Migrer ou répliquer
- Pas de cohérence de cache matérielle
  - Empêcher accès concurrents distants
    - Migrer sans répliquer
      - Sauf si accès uniquement en lecture

Différents problèmes

- DSM matérielle
  - Illusion matérielle d'une grosse NUMA
  - Mêmes problèmes que machines NUMA
    - Migration utile pour performance
    - Pas indispensable
- DSM logicielle
  - Illusion logicielle
  - Migration indispensable avant accès distant
    - **Enorme** coût en performance
  - Maintien manuel de la cohérence

Facteur NUMA et migration de pages

- Accès direct aux pages distantes
  - Gros facteur NUMA
    - Au moins 1us sur NUMAlink au lieu de 100ns en local
- Migrer les pages là où elles sont utilisées
  - Affinité mémoire
    - Comme pour les caches
- Nécessité de bloquer les accès pendant la migration

Impact sur l'ordonnancement

- Plus l'accès distant est lent, plus il faut l'éviter
  - Migrer les processus près des données
  - Migrer les données près des processus
- Contraintes sur l'ordonnancement proportionnelles au coût réseau
  - Critique si latence élevée et/ou débit faible

Réplication

- Si aucune écriture
  - Utiliser la mémoire locale comme un cache
    - Tant qu'il y a de la mémoire disponible
- Répliquer les données dans la mémoire locale de tous les nœuds au début du programme
  - Exploitation localité temporelle et spatiale
  - Exploitation parallélisme
  - Application insensible au placement initial des données

Accès concurrents

- Si plusieurs processus accèdent aux mêmes données
  - Géré par la cohérence de cache en local
  - Comment en faire en distribué ?
    - Interdire les accès concurrents ?
    - Réplication uniquement en lecture seule

Cohérence de cache

- Les caches d'une machine sont similaires aux mémoires des machines d'une DSM
- Cache mémoire critique pour performance
  - Efficacité maintien cohérence est critique pour performance
    - Protocole complexe dans le matériel
- Machines *Cache Coherent*-NUMA

Cohérence de cache (2)

- Machines non-CC-NUMA
  - Très difficile à programmer
    - Disparaît peu à peu
      - Mais pourrait bien revenir...
        - Intel SCC, ...
  - Soit pas de cache local
  - Soit pas de cache sur données partagées
    - Accès lents
    - Similaire à SCI

Consistence et cohérence

- Cohérence = ordre des accès à une variable
  - Lecture après écriture retourne la dernière valeur
- Consistence = ordre global de tous les accès
  - Dépendences entre accès à variables différentes
  - Un exemple où la cohérence ne suffit pas ?

Consistence et cohérence (2/2)

- Modèles de consistence mémoire
- Protocoles de cohérence (caches, ...)
- On parle souvent de cohérence par abus de langage

Consistence

- Consistence séquentielle
  - Toute modification est **immédiatement** visible sur les autres nœuds
- Consistence relâchée ou faible
  - Pas immédiatement visible
    - Exemples de garanties
      - Délai de propagation
      - Dépendances sur certaines opérations spéciales
        - Prise/relâchement verrou propage les modifications

Granularité

- Octet
  - Accès transparent
    - Très simple à utiliser
  - Très complexe à mettre en place
- Page
  - Plus simple à mettre en place
  - Faux partage
- Objets partagés
  - Faux partage en interne, éventuellement
- Ligne de cache
  - Utilisé dans le matériel au lieu de l'octet

Propagation des modifications

- *Write-Update*
  - Propagation immédiate des modifications locales
  - Nécessite beaucoup de messages et de synchronisation
- Bien si beaucoup d'accès concurrents
  - Beaucoup de gens vont avoir besoin des modifications, autant leur envoyer tout de suite
  - Pas la peine de maintenir la liste précise des copies, elle est trop longue
    - Broadcast des modifications

Propagation des modifications (2/2)

- *Write-Invalidate*
  - Invalidation préalabre des copies distantes
  - Propagation quand accès distant
  - Permet plusieurs modifications locales avant propagation
- Bien si peu d'accès concurrents
  - Pas besoin de propager immédiatement si personne n'en a besoin immédiatement
  - Nécessité d'invalider efficacement
    - Maintenir la liste exacte des copies distantes
      - Elle est censée être petite

Modèle de Ivy

- Basé sur les pages
- Cohérence séquentielle
- État global d'une page
  - Lecture par plusieurs nœuds
  - Lecture ou écriture par un seul nœud
- Garantie d'une cohérence séquentielle
  - Nouvel accès en lecture rapatrie copie locale
    - Réplication
  - Nouvel accès en écriture bloque tout le monde
    - Invalidation

Modèle de Ivy (2)

- État local possible d'une page ?
  - *Invalide*
  - *Lecture seule partagée*
    - Invalide ou lecture seule sur les autres nœuds
  - *Lecture/écriture exclusive*
    - Invalide sur les autres nœuds
- Changement d'état provoqué par les types d'accès sur les différents nœuds

Modèle de Munin

- Basé sur des objets
- Cohérence relâchée
- Objets de synchronisation manipulés par l'application pour aider la DSM
  - Pas transparent
- Cohérence uniquement assurée quand on relâche un objet et qu'un autre l'acquiert

Le protocole MSI

- *Modified*, *Shared*, *Invalid*
- Conçu pour cohérence entre caches matériels
  - Implémenté en hard dans les processeurs et bus mémoire
    - Avec améliorations
  - Applicable aux DSM
    - Pages ou objets de la DSM au lieu des lignes de cache

Le protocole MSI (2/2)

- Signification des états ?
- Automate de transition entre état ?
  - Lecture ou écriture, locale ou distante

L'extension MESI

- Ajout de l'état *Exclusive*
  - Éviter le coût de la transition S->M
    - Pourquoi ?
    - Que devient l'automate de transition ?

Autres extensions

- Etat *Owned* (AMD)
  - Permet de déferrer la mise à jour de la mémoire centrale
    - Évite le coût d'accès à la mémoire centrale quand l'accès aux autres caches est plus rapide
      - Décision dynamique entre Modified+Invalid ou Owned+Shared
- Etat *Forward* (Intel récent)
  - Si une ligne est en S, il y a **une** copie en F
  - Le propriétaire du F se charge de répondre aux demandes

Les protocoles MESI* pour les DSM

- Objets DSM au lieu des lignes de cache
- État *Owned* a priori inutile dans DSM
  - Pas de mémoire centrale derrière
    - Ou ce sont les disques ?
- État *Forward* utile
  - Il faut un responsable par page
- Peut nécessiter de nombreuses diffusions à tous les possesseurs d'une page
  - Broadcast réseau cher
  - État *Exclusive* très utile

Gestion des pages

- Répertoires d'attributs
  - Localisation des pages virtuelles
  - Protection des différentes copies physiques
- Un gestionnaire est contacté pour chaque changement d'état

Gestionnaire centralisé

- Un nœud chargé de traiter les demandes
- Contacte le(s) propriétaire(s) de la page
- Donne une copie au demandeur
- Met à jour le répertoire global
  - Nouvel état et propriétaire
- Tout ceci doit être protégé contre les accès concurrents

Gestionnaire distribué dynamique

- Gestionnaire = propriétaire
- Maintien d'un propriétaire probable
  - Propagation des requêtes sur une chaîne de propriétaires probables jusqu'à trouver le bon
    - Mise à jour du propriétaire probable sur le demandeur et l'ancien propriétaire
  - Quel est le cas le pire ?
- Traitement des pages une par une par le gestionnaire local sur le propriétaire correspondant

Stockage des attributs globaux

- 1 octet pour le statut global
- Quelques octets pour liste des copies
  - Si trop de copies, ne pas garder la liste précise ?
- Occupation mémoire proportionnelle à nombre total de pages de tous les noeuds
  - Répartie sur tous les noeuds si gestionnaire distribué dynamique
    - En moyenne, proportionnelle à mémoire d'un noeud

Stockage des attributs locaux

- 1 octet pour le statut local et/ou le propriétaire probable
- Occupation mémoire proportionnelle à la mémoire totale de tous les noeuds
  - Peut-être problématique
    - Oublier le propriétaire probable des pages non récemment accédées (LRU)
- Et si ca occupe trop de mémoire ?
  - Augmenter la taille des pages ?

DSM gérée dans l'application

- Contrat entre utilisateur et la DSM
  - Permet de réduire les garanties
    - Modèle de consistence
    - Plus facile à mettre en œuvre
- Espace d'adressage virtuel spécifique
  - Données identifiées par offset
  - Primitives d'accès spécifiques
    - val = Lit(addr) au lieu de val = *addr
    - Ecrit(addr, val) au lieu de *addr = val

DSM dans un *Middleware*

- Assez générique, moins adapté à l'application
- Impose garanties bien définies
  - Modèle de consistence
- Plus complexe à mettre en œuvre
  - Doit détecter et réagir aux accès concurrents
    - Comme dans l'OS, avec moins de support logiciel
- Peut proposer différentes consistences à l'application

Défaut de page en espace utilisateur

- Appel système mprotect() pour rendre zone *Shared*, *Exclusive* ou *Invalid*
- Rattrapper le signal SEGV
  - Migrer ou recopier les données
  - Remettre la protection normale

DSM dans l'OS

- Transparent pour l'utilisateur
  - Déréférencement habituel des pointeurs
- Générique
  - Difficile à adapter à application quelconque
    - Heuristiques pour deviner ce que l'application fait
    - Assistance de l'application
- Utiliser les protections mémoire
  - Détecter un accès en lecture
    - Rapatrier une copie de la page
  - Détecter un accès en écriture
    - Récupérer un droit sur la page

Implémentation des défauts de page dans l'OS

- Page invalide
  - L'OS alloue une page libre
  - Si page locale non partageable
    - Lire sur le disque
    - Ajuster la protection
  - Sinon
    - Contacter le gestionnaire
    - Rapatrier la page depuis le réseau
    - Ajuster la protection
      - Lecture seule si accès en lecture

Implémentation des défauts de page dans l'OS (2)

- Accès en écriture dans une page en lecture seule
  - Si page locale non partageable
    - Si Copy-on-write
      - Dupliquer et autoriser l'écriture
    - Sinon *Segmentation fault*
  - Sinon
    - Contacter le gestionnaire
      - *Segmentation fault* si page en lecture seule
    - Rapatrier page si nécessaire
    - Autoriser l'écriture

Implémentation des défauts de page

- Évaluation des coûts ?
- Nécessité de réfléchir à l'implémentation du protocole de cohérence
- ... et aux algorithmes

Faux partage

- Illusion de données partagées
  - Ping-pong entre 2 nœuds
    - Coût ?
- Comme dans les caches en local
  - Granularité différente
    - Zone quelconque (page?) au lieu de lignes de cache
  - Maintien de la cohérence
- Comme dans les machines NUMA
  - Migration au plus près

Gestion du faux partage par *Copy-on-Write*

- Détecter les modifications locales par *Copy-on-Write*
  - Garder l'original inchangé
  - Faire un diff entre l'original et la copie modifiée
- Envoyer le diff aux autres propriétaires d'une copie la page
  - Ils vont essayer de fusionner les changements
- Utiliser ceci pour le faux partage
  - Et synchronisation sur les zones en vrai partage
- Impose de savoir à l'avance les zones en vrai et faux partage

Double défaut de page

- Accès en lecture puis accès en écriture
  - Courant quand on modifie une valeur
  - D'où l'extension MESI du modèle MSI
- Difficile à éviter si la DSM est basée sur des défauts de pages matériels
  - Défauts causés par instructions exécutées par le processeur
- Facile à traiter dans le cas de DSM gérée par l'application
  - Adapter l'interface d'accès à la DSM pour éviter le problème (lecture et écriture simultanées ?)

Anticiper

- Prévoir les prochains accès d'un nœud
  - Localité spatiale
  - Regroupement des accès
- Facile dans une DSM dans application
  - L'application peut prévenir elle-même
- Détection difficile dans les DSM dans l'OS
  - Comme le *Read-ahead* des fichiers
    - Avec accès concurrents

Communications lors d'un défaut de page dans une DSM

- La tâche doit rester bloquée en attendant
  - Et les autres threads ne peuvent pas toucher à la même zone
    - Faire très attention aux accès concurrents
- Envoi de requête puis bloquer en attendant réponse (migration de page)
  - Modèle client-serveur simple

Comment transférer les données dans une DSM ?

- Synchronisation explicite lors d'un rapatriement de page
  - Requête + attente Réponse
  - On peut transférer par RDMA entre les 2
    - Destination mémoire précisée dans requête
- Moins de synchronisation si migration par anticipation
  - Mais nécessité de prévenir quand la migration est terminée
    - Problème très similaire au final

Communications lors d'un changement d'état

- Si lecture dans zone invalide
  - Trouver la page et rapatrier une copie
- Si écriture dans zone en lecture seule
  - Pas de migration nécessaire
- Invalider tous les autres
  - Rien à faire si on passe de *Exclusive* à *Modified*
  - Prévenir tout le monde ?
    - Comment faire ça efficacement ?
- Attendre la confirmation d'invalidation

Comment invalider les copies distantes ?

- Connaître la liste des copies
  - Coûteux en mémoire?
    - Dépend granularité et taux accès concurrents
  - Ajoute des communications
    - Prévenir les autres lors d'une nouvelle copie
  - Réduit communications lors de l'invalidation
  - Bien si beaucoup d'invalidations et peu de copies
- Prévenir tout le monde, même si inutile
  - Bien si beaucoup de copies et pas trop d'invalidations

Comment invalider les copies distantes ? (2/2)

- Comment invalider toutes les copies ?
  - Broadcast réseau coûteux ?
  - Multicast ou collectives MPI imposent de peu changer l'ensemble des destinataires
  - Une requête par destinataire
    - Pipeliner pour réduire le coût

Comment traiter les requêtes des autres dans une DSM ?

- L'application passe son temps à calculer
  - Traitement des besoins locaux pendant défauts de page (traitants de signaux)
- Requêtes inattendues des autres
  - Besoins distants difficiles à prévoir
    - Sauf si anticipation par heuristiques
  - Pouvoir traiter requête à tout moment
    - Thread dédié aux communications entrantes?
    - Modèle actif?
      - Souvent basé sur des threads en dessous

Synchronisation distribué vs. DSM?

- Variables de synchronisation stockées dans la DSM?
  - Un spinlock serait extrèmement couteux
    - Ping-pong de pages
  - Déjà dans une machine NUMA, ça coûte cher
- Utiliser synchronisation par communication/messages explicites
  - Plus de travail pour le développeur mais beaucoup plus efficace

Rappels sur la synchronisation locale

- Mutex
- Sémaphore
- Spinlock

- Basé sur des opérations atomiques
  - test_and_set
  - cmpxchg
- Très lent sur certaines architectures

Généralisation à la synchronisation globale ?

- Opérations atomiques sur une DSM ?
  - Impose cohérence forte
- Coûteux sur une DSM hardware
  - Latence des accès à la mémoire distante
- Très dur et lent sur DSM logicielle
  - Ping-pong de pages
- Trouver d'autres méthodes de synchronisation
  - Passage de messages explicite

Retour sur les coprocesseurs

- Cell, GPGPU, ... ont mémoire embarquée
  - Les périphériques n'ont pas forcément d'accès direct à la mémoire centrale
    - DMA nécessaire, coûteux
  - Le processeur central n'a pas forcément accès à toute la mémoire des périphériques
    - Même si PIO possible, c'est coûteux

Migration de données entre (co-)processeurs

- Nécessité de migrer les données à coté du processeur qui les utilise
  - Migrer par gros blocs pour masquer la latence et profiter du débit élevé
    - Centaines de nanosecondes et Go/s
- Pas de défauts de pages
  - Migration explicite des données
    - Déterminer les données nécessaires et les migrer avant de lancer la tâche sur le coprocesseur
  - Granularité libre

Systèmes d'exploitation à image unique

Présentation

- SSI (*Single System Image*)
- La grappe est la machine parallèle du pauvre
  - Corriger cela logiciellement
- Mise en commun et partage des ressources de multiples machines physiques
- Exposition de ressources logiques de manière uniforme sur tous les nœuds

Types d'applications parallèles

- Tâches parallèles fixes
  - Nombre fixe de processus
  - Fortement synchronisés
  - Allocation du nombre exact de processeurs
- Tâches dynamiques
  - Ajout/suppression de processus
  - Répartir la charge

Objectifs des SSI

- Flexibilité
  - Nombre de processeurs adapté
  - Utilisation identique quelle que soit la taille de l'application
- Permet programmation entièrement par threads et mémoire partagée sur multiples machines physiques
  - Pas besoin de MPI
    - Si c'est performant...

Transparence

- Accès aux ressources logiques depuis n'importe quel nœud
  - Fichiers locaux ou distants
- Migration de ressources logiques sans changer la méthode d'accès ni perturber l'exécution
  - Espaces de nommage

Extensibilité

- Augmentation performances globales avec augmentation des ressources mises en commun
- Reconfiguration dynamique
  - Ajout/suppression de nœuds

Extensibilité (2)

- Décentralisation
  - Pas de goulet d'étranglement
  - Tolérance aux pannes et disponibilité
- Réplication
  - Différentes copies d'une même donnée réparties et gérées par le système

Différences avec DSM

- SSI contient souvent DSM dans l'OS
- SSI partage aussi d'autres ressources
  - Périphériques
- Migration de tâches entre machines
  - DSM laisse tâche au même endroit et migre données si nécessaire

Migration de tâches

- Répartir la charge
  - Profiter des machines inactives
- Disposer des ressources non-partageables
  - Co-processeur
  - Carte accélératrice
  - ...

Migration de tâches (2)

- Placer correctement les entrées-sorties
  - Processus accédant aux données
    - Près des disques
  - Processus communicants entre eux
    - Près l'un de l'autre
  - Si les I/O saturent
    - Répartir les processus

Quand et quoi migrer ?

- Lors d'un accès à ressource spécifique
  - Migrer le processus fautif près de la ressource
- Quand le système le décide
  - Équilibrage de charge
    - Politique de décision du nœud cible
- En cas de défaillance
  - Restaurer les processus morts
- Quand c'est possible
  - Processus qui utilise uniquement des ressources partageables

Préparer la migration

- Geler le processus dans un état exploitable par la machine distante
  - Pas au milieu d'un appel système
    - Aucune ressource locale en cours de modification
  - Attendre fin des traitements en cours
    - Communications, accès disques, ...
  - Références aux données système partagées doivent être références globales
    - Encapsulation des ressources physiques

Contraintes sur la migration

- Architecture des machines
  - Identique ou non (PGCD des processeurs)
  - SMP ?
  - Plusieurs binaires multi-architecture dans le même exécutable ? Recompilation à la volée ?
    - Comment sauver/restaurer les registres entre différentes architectures ?
      - Insérer des points de sauvegarde via le compilateur ?

Migration concrète

- Migration de toutes les pages mémoire
  - En utilisant la DSM du système distribué
- Migration du contexte
  - Sauver/restaurer comme lors d'un changement de contexte
- Processus interrompu pendant longtemps
  - Comment optimiser ?

Optimiser le temps de migration

- Pré-copier les pages mémoire avant le gel
  - Très utile pour pages en lecture-seule
    - Code et certaines données
  - Possible pour pages en lecture-écriture
    - Recommencer la copie si nouvelle modification avant le gel
    - Trouver un compromis
      - Utiliser les informations du cache de pages
        - LRU, FIFO, ...
  - Charge la machine destination plus tôt
  - Réduit le temps de migration *apparent*

Optimiser le temps de migration (2)

- Copier les pages de manière paresseuse
  - Ne migrer que quand c'est nécessaire
    - En profitant de la DSM
  - Charge la machine source plus tard
  - Réduit le temps de migration *apparent*

Reprise après migration

- Si la machine destination n'a pas les ressources nécessaires
  - La machine source reprend l'exécution
- Sinon, acquittement
  - La machine source oublie le processus migré

Reprise après migration (2)

- Rapatrier les ressources logiques nécessaires
  - Les relier localement aux ressources physiques
    - Pages mémoire, fichiers, sockets, ...
- Restauration du contexte
  - Comme lors d'un changement de contexte

Reprise sur erreur

- Migration permet sauvegarde état d'un processus
  - Permet de reprendre l'exécution au même droit plus tard
- En cas de problème, une sauvegarde permet de reprendre l'exécution
  - Mort d'une des machines
    - Processus transférés et relancés sur une autre
- Nécessite support de stockage fiable et performant

Reprise sur erreur (2)

- Sauvegarde de l'état périodiquement
  - État vis-à-vis du système
  - État au niveau de l'application
    - Communications en cours
- L'application se charge de sauver quand elle le juge nécessaire
  - En collaboration avec les autres nœuds
    - Pour disposer d'un état global sauvegardé
  - Pas forcément tous les nœuds en même temps
    - Pour ne pas saturer le système d'entrées-sorties

Accès mémoire dans les SSI

- DSM au cœur du système
- Partage mémoire entre processus
  - Dépend du SSI
  - Si DSM complète avec cohérence forte
    - Partage mémoire normal
  - Sinon, on peut imposer des contraintes, ex :
    - Threads placés sur même machine physique
    - Pas de partage mémoire entre processus
    - Partage mémoire en lecture
    - Partage mémoire en écriture sans cohérence forte

Accès transparent aux fichiers

- Espace de nommage global
  - Système de fichiers distribué
- Migration de pages dans le cache global de fichiers
- Migration des descripteurs de fichiers

Accès transparent aux périphériques

- Périphériques logiques virtualisés
  - Migration de l'encapsulation
- Périphériques physiques
  - Gestionnaire local relié aux processus distants l'utilisant par des connexions réseau

Accès transparent au réseau

- Bloquer une socket pendant la migration
  - Saturation possible
    - Données seront ré-émises automatiquement
- Transférer données accumulées vers nouvelle machine
- Mettre en place socket entre nouvelle machine et la relier aux destinataires
- Même problème avec les tubes

Accès transparent au réseau (2)

- Forwarder une socket ?
  - Tunnel passant par la machine précédente pour faire suivre les données à la nouvelle
  - Comment réagir aux migrations successives ?
- Migrer une socket ?
  - Même adresse IP sur toutes les cartes réseau
    - Problème de routage
  - Changer l'IP associée à la socket ?
    - Problème de connexion

Gestion des cartes réseau

- Cartes réseau utilisées pour
  - Communications dans le SSI
    - Adresse IP privée ?
  - Communications entre le SSI et l'extérieur
    - Adresse IP globale ou multiples ?
- Réserver chaque carte pour un seul usage ?
- Filtrer les paquets SSI ou normaux ?

Gestion des processus

- Nécessité d'identifiants globaux
  - Signaux
  - Relation père-fils
- Table globale de PID encapsulant des processus locaux bas-niveau

Dépendances résiduelles

- Multiples structures localement inutiles
  - Pages non-migrées (migration paresseuse)
  - Périphériques physiques exportés par un gestionnaire local
  - Connexions réseau forwardées
- En tenir compte lors du choix de la destination d'une migration
- Processus peut être affecté par défaillance d'une autre machine

Mise en œuvre en espace utilisateur

- Développement assez facile
  - Mais peu de possibilités
  - Transparence limitée
  - DSM limitée
- Suffisant pour les problèmes non-critiques
  - Politiques de migration
  - Services de gestion du SSI
    - Découverte des machines

Mise en œuvre dans le noyau

- Développement complexe
  - Très intrusif
  - Difficile à maintenir et faire évoluer
- Très flexible
  - Transparence totale
    - Virtualisation de ressources à travers l'interface standard
  - DSM à cohérence forte possible
    - Défauts de page

Parallel Virtual Machine

- Système intégré de communication et migration de processus
  - Pour les applications parallèles
- Démon local sur le nœud
  - Détecte surcharge locale
  - Contacte un nœud libre
  - Migre la tâche et adapte les communications
- Mise en œuvre en espace utilisateur

OpenMosix

- Version Open-Source de Mosix
  - *Multicomputer Operating System for unIX*
- Système d'exploitation distribué
  - Patch pour le noyau Linux
- Ajout/retrait de nœuds à chaud
  - Pas de reprise sur erreur
- Très stable mais fonctionnalités limitées
- Fin du projet annoncée récemment

OpenMosix (2)

- Système probabiliste de détection et équilibrage de charge
  - Chaque nœud connaît une approximation de la charge des autres
  - Passe bien à l'échelle
- Migration du corps des processus
  - Modèle du *home-node*
    - Mandataire reste sur nœud initial
      - Contexte noyau
      - Traitement des appels-système
  - Lien persistant entre les deux

OpenMosix (3)

- Pas de vision globale des processus
  - Utilitaire spéciaux dédiés
- Pas de mémoire partagée distribuée
  - Pas de migration de threads
  - Pas de migration de processus partageant de la mémoire
- Migration des processus manipulant les sémaphores, sockets, tubes ou fichiers

OpenSSI

- Système à image unique
  - Patch pour le noyau Linux
- Beaucoup de fonctionnalités
  - Mais limité en performance
    - Communication inter-processus
- Ajout/retrait de nœuds à chaud
  - Tolérant aux pannes, sauf sur le nœud racine

OpenSSI (2)

- Vision globale des processus et des périphériques avec les outils standards
- Support des méthodes de communications inter-processus
  - Mémoire partagée
  - Sémaphores
  - Sockets et tubes
- Pas de vision globale de la mémoire
- Pas de migration de threads individuels

Kerrighed

- Système à image unique
  - Patch pour le noyau Linux
- Beaucoup de fonctionnalités et performant
  - Support des machines SMP encore en développement
  - Pas d'ajout/retrait de nœuds à chaud, ni de tolérance aux pannes
- Version stabilisée en cours de développement

Kerrighed (2)

- Système à image unique
  - Migration de tâches
    - Y compris threads individuels
    - Outils non-standards pour les manipuler
  - Mémoire partagée distribuée
    - Vision globale de la mémoire par outils standards
  - Reprise sur erreur pour programmes séquentiels
  - Système de fichiers parallèle

Kerrighed (3)

- Notion de *Container*
  - Encapsulation migrable de ressource virtualisée
    - Fichier
    - Segment de mémoire partagée
    - ...
  - Cohérence séquentielle en cas de partage
  - Gestion des copies multiples
  - Base de la gestion du système global

Conclusion

- La plupart des fonctionnalités sont supportables
  - Gros travail de développement et de maintenance dans le noyau
  - Mémoire partagée distribuée contraignante à mettre en place mais utile
  - Vue globale des ressources (notamment mémoire et processus) assez aisée à mettre en œuvre

Virtualisation ?

- Migration de machines virtuelles pour tolérer les pannes
  - Migration transparente des processus dedans
- Pourquoi ca marche mieux ?
  - Xen connecté par *Bridge Ethernet*
    - Mettre à jour le switch et ca marche
      - Ping ARP, ...
    - Les connexions IP suivent

Stockage distribué

Un domaine souvent étudié...

- FTP, NFS, Coda, AFS, InterMezzo, PVFS, GPFS, Lustre, Samba, NCPFS, CIFS, DAFS, XFS, CFS, Sprite, HPSS, GFS, DCE/DFS, Cosmos, NFSp, pNFS, DOMAIN, PPFS, iSCSI, AoE, KerFS, GmailFS, GoogleFS, ...
- La roue a été réinventée plusieurs fois...

Généralités

- Ensemble de données qui doit être accessible depuis tous les nœuds du système
  - Pas uniquement les disques locaux pour les processus locaux
- Accès aux fichiers non transparent
  - FTP (*File Transfer Protocol*)
    - Peu intuitif
    - Pas transparent

Objectifs

- Accès transparent depuis tous les nœuds
- Illusion d'un seul système de stockage
- Utiliser tous les disques disponibles
- Supporter la charge imposée par les clients
  - Stockage = Facteur limitant la performance
- Exploiter la mémoire des nœuds comme un cache de fichiers global

Différents besoins

- Applications régulières
  - Même application sur différents nœuds et volumes de données
    - Schéma d'accès aux données connus à l'avance
- Applications irrégulières
  - Concurrence
  - Schéma d'accès aux données inconnu
- Applications *Out-of-core*
  - Données ne tenant pas en mémoire
    - Va-et-vient entre stockage et mémoire

Différentes réponses aux besoins

- Stockage distant distribué à différents clients
  - Espace de nommage global
  - Accès transparent depuis n'importe quel client
- Stockage caché dans le client
  - Réduire la charge de travail du serveur
  - Réduire les aller-retours sur le réseau
  - Améliorer les performances locales

Différentes réponses aux besoins (2)

- Stockage réparti ou parallèle
  - Support de la charge des clients
  - Réaction à la distribution géographique
  - Essentiel dans les grappes de calcul
    - PVFS, Lustre, ...
- Stockage répliqué
  - Tolérance aux pannes
  - Essentiel pour les systèmes répartis géographiquement
    - AFS, Coda

Différents niveaux de mise en œuvre

- Bibliothèque utilisateur
  - Manipule des chemins, descripteurs et octets
- Système de fichiers distribués
  - Manipule des i-nœuds et octets
- Cache de fichiers distribué
  - Manipule des i-nœuds et pages

Différents niveaux de mise en œuvre (2)

- Système de blocs distribués
  - Manipule des blocs logiques
- Stockage local ou attaché au réseau
  - Manipule des blocs physiques

Transparence

- Accéder aux données distantes comme si elles étaient locales
  - « montage »
  - Interconnexion des espaces de nommage local et distant
    - Remplacer un répertoire existant par un répertoire distant et son contenu
    - Transition automatique entre les 2 espaces

Mise en œuvre de la transparence

- Couche logicielle de virtualisation
  - Plus facile à mettre en œuvre dans l'OS
- Chaque type de fichier a ses méthodes d'accès
  - Fichiers locaux
    - Accès aux disques locaux
  - Fichiers distants
    - Requêtes aux serveurs distants à travers le réseau

Conservation de l'état

- Dans le serveur
  - Lien étroit entre client et serveur
  - Difficulté de passage à l'échelle
  - Pas de tolérance aux pannes du serveur
- Dans le client
  - Mode « déconnecté »
    - Client pas informé des modifications concurrentes
    - Problème de cohérence

Représentation des fichiers distants

- Nom serveur + identifiant local
  - Serveur central
    - Goulet d'étranglement
    - Pas de migration du stockage
- Identifiant global
  - Localiser le serveur
    - Serveur de noms
      - Cache de noms facile à mettre en place

Cache de fichiers

- Conserver données en mémoire
  - Réduire les accès disques
    - Latence importante et débit limité
  - Améliorer la vitesse de lecture
  - Réduire la durée apparente des écritures
    - Écritures physiques déférées en tâche de fond
  - Utilisation de toute la mémoire disponible
    - En régime permanent, il n'y a jamais de mémoire disponible sur un système moderne

Différents niveaux de cache

- Cache de noms
  - Localiser des fichiers
- Cache de pages sur le client
  - Réduire les accès au serveur distant
- Cache de pages sur le serveur
  - Réduire les accès au matériel de stockage
- Cache matériel
  - Réduire les accès aux périphériques mécaniques

Cache coopératif

- Mettre en commun les caches de différents nœuds
  - Similaire à GMS pour les fichiers
- Accès réseau plus rapide qu'un accès disque
- Peut être un sur-cache au cache local
  - Rend complexe l'accès aux pages des caches non-locaux

Cache coopératif glouton

- Les nœuds remplissent leur cache indépendamment les uns des autres
- En cas de défaut de page
  - Le nœud contacte le serveur
  - Le serveur sait si des nœuds ont la page
    - Le serveur envoie lui-même la page si aucun
    - Le serveur fait suivre la demande à un nœud qui a cette page
      - Le nœud envoie la page à l'initiateur
- Goulet d'étranglement

Cache coopératif statique

- Cache associatif global contenant les ensembles des caches locaux des nœuds
  - Fonction de hachage pour déterminer le nœud gérant la page
- Pas de réplication
  - Pas de problème de cohérence
- Contacter le nœud responsable pour chaque accès
  - Trafic réseau
  - Mauvaise localité spatiale et temporelle

Cache collaboratif centralisé

- Compromis entre glouton et statique
- Chaque nœud contient
  - Cache local privé
  - Cache global = extension cache du serveur
- Défauts de pages gérés comme dans glouton

Cache collaboratif centralisé (2)

- Si mémoire serveur saturée, évincer pages dans cache global sur autres nœuds (LRU)
  - Taux de succès élevé dans cache global
    - Variable dans cache local
- Goulet d'étranglement important

Cache collaboratif distribué

- Cache glouton avec copies multiples
  - Blocs répliqués évincés en priorité
    - Puis LRU
  - Notion de *singleton*
    - Favorisés
- Blocs cachés dans nœuds qui ont grande probabilité de les utiliser
- Élimination des blocs migrés plusieurs fois sans être utilisés

Cache collaboratif distribué (2)

- Mécanisme de cohérence complexe
- Réplication implique sous-utilisation de la mémoire pour le cache
- Gain en performance par localité
  - Mais beaucoup de messages échangés
- Mis en œuvre dans XFS

Cohérence dans le stockage distribué

- Nécessaire si
  - Cache dans le client et accès concurrents
  - Serveur parallèle ou réparti
  - Stockage répliqué
- Granularité de la cohérence dépend de la structuration des données
  - Blocs
  - Fichiers
  - Pages ou segments de fichiers

Sémantique de cohérence Unix

- Modification d'un fichier immédiatement vue par les autres applications
- Ne pas cacher les écritures (Sprite)
  - Prendre un jeton avant écriture
  - Ne pas accéder au fichier si quelqu'un tient un jeton
    - Et invalider le cache
  - Très lent
- Expiration du jeton nécessaire pour tolérer les pannes

Adapter la cohérence aux besoins

- Maintien cohérence réduit performance
  - Adapter la cohérence aux besoins des applications
    - NFS pour les homes
      - Pas d'accès concurrents
      - Performance du cache suffisante
    - PVFS pour les applications parallèles
      - Pas d'accès concurrents au même segment de fichiers
      - Performances importantes

Relâcher la cohérence Unix

- Modifications visibles immédiatement pour les applications locales
  - Sémantique Unix respectée dans cache local
- Même problème que dans une DSM
  - Mais plus de possibilités
- Structure et granularité des données
  - Faux accès concurrents
    - Fichiers différents
    - Segments de fichiers différents

Sémantique *Close-Open*

- Modifications propagées à la fermeture du fichier
  - Un client sera à jour s'il ouvre le fichier juste après
- Propagations périodiques
  - Pas de garantie précise
- NFS, AFS, ...

Écritures seules concurrentes

- Propagation synchronisée par tous les clients à intervalle régulier
  - Quand un client a besoin de mémoire
- Conservation des dates de modification pour choisir le plus récent
  - Nécessite horloges à peu près synchronisées
- Modifications perdues en cas de faux partage

Projection de fichier

- *Mapping* d'un segment de fichier en mémoire virtuelle
  - Accès aux fichiers comme dans une DSM
    - Pas de lecture/écriture explicite
    - Déréférencement de pointeur
  - Adapter protocole de cohérence
    - Refuser projections partagées

Chargement à l'avance

- Éviter les prochains défauts dans le cache des fichiers
- Regrouper les accès disques
- Pas trop tôt
  - Sinon gaspillage mémoire et ralentissement
- Pas trop tard
  - Sinon attente de la fin du chargement

Chargement adapté aux besoins

- Heuristiques pour deviner comment précharger
  - Accès séquentiel ou aléatoire ?
  - Indications de l'utilisateur
    - fadvise()
- Préchargement des caches de différents nœuds
  - Prévoir quel nœud va utiliser quelle partie d'un fichier

Accès non-cachés

- Flag O_DIRECT à l'ouverture du fichier
- Lecture et écriture sans passer par le cache
- Accès lents à travers le réseau

Accès parallèles

- Applications parallèles traitent gros volumes de données
  - Tous les nœuds lisent et écrivent dans le même système de stockage
  - Même fichier ou fichiers distincts
- Pointeur de fichier partagé
  - Accès séquentialisés
    - Lent

Accès parallèles (2)

- Définition de segments de données spécifiques à chaque nœud
  - Structure/granularité des données à traiter dans l'application parallèle
    - Souvent prévisible à l'avance
- Adaptation du stockage à cette segmentation

Accès parallèles (3)

- Accès par pas
  - Ensemble de segments de taille fixée à intervalle régulier
- Accès par pas imbriqués
  - *Pattern* ou *Stride* à 2 niveaux
- Définition d'un type complexe
  - Passé à l'interface de programmation des accès au système de stockage
    - MPI-IO, Vesta, ...

MPI-IO

- Extension MPI-2 pour les entrées-sorties
- Interface utilisateur
  - Accès au stockage
  - Spécification de *Patterns*
- Conçu pour applications parallèles
  - Correspond aux patterns usuels
  - Orienté performance

MPI-IO (2)

- Définition des éléments/types de base
- Ouverture des fichiers
  - Spécification de sous-parties
  - Affectation des sous-parties aux processus
- Accès lecture/écriture
  - Spécification des emplacements mémoire où lire/écrire les données

Fragmentation du stockage

- Répartition des données sur différents serveurs
- Distribution de la charge
  - Serveur centralisant la gestion de la fragmentation
    - Éventuellement serveur distribué
- Pas de problème de cohérence
- RAID 0, LVM, PVFS, ...

Réplication du stockage

- Tolérance aux pannes
- Distribution géographique
  - Être près de tous les clients
- Service centralisé
  - Pas de problème de cohérence
  - Goulet d'étranglement possible
  - RAID 1, LVM, ...

Réplication du stockage (2)

- Serveurs indépendants
  - Pas de synchronisation immédiate
    - Performances non-limitées
  - Reconnexion et synchronisation régulière
  - Fusion des modifications
    - Dernière instance modifiée du fichier
      - Pas de fusion interne au fichier
      - Coda, AFS, ...

RAID

- *Redundant Array of Inexpensive Disks*
- Assemblage de disques multiples pour créer de gros disques logiques fiables
  - Matériel
    - Rapide
  - Logiciel
    - Assez rapide et flexible
      - Distribué géographiquement
- Performances, capacité, tolérance aux pannes, extensibilité

Fragmentation (RAID-0)

- *Striped* RAID
- Agrégation de disque
- Blocs stockés en alternance sur chacun des disques
- Granularité élevée (64ko)
  - Profiter du débiter élevé
  - Recouvrir les temps d'accès
- Pas de tolérance aux pannes

Réplication (RAID-1)

- *Mirrored* RAID
- Disques identiques contenant les mêmes données
- Remplacement d'un disque défectueux
- Réparation en tâche de fond
- Perte de 50% de la capacité totale

Détection des fautes (RAID-3/4)

- Disques dédiés aux bits de parité
  - RAID 3 ou RAID 4
  - Plus de disques de données que de disques de parité
    - Moins de 50% de perte de capacité
  - Agrégation de capacité
  - Tolérance aux pannes
  - Coûteux si mis en œuvre logiciellement

Détection des fautes (RAID-5/6)

- Bits de parité distribués en *Round-Robin* parmi les données
  - RAID 5 ou RAID 6
  - Agrégation de capacité
    - Peu de capacité perdue
  - Tolérance aux pannes
  - Coûteux si mis en œuvre logiciellement

Stockage parallèle

- Différents niveaux de RAID combinables
  - RAID 10, 15, 50, ...
- Mise en œuvre dans serveurs parallèles pour les grappes
  - PVFS, ...
- L'empilement de toutes ces couches permet de créer du stockage distribué quelconque
  - Réplication, répartition, passage à l'échelle, cohérence, ...

Accès au stockage par blocs

- Données non structurées
  - Pas de fichiers
  - Pas de répertoires
  - Cohérence au niveau des blocs
- Verrou global
  - GFS, GPFS, ...
- Stockage des blocs sur
  - Disques locaux
  - *Network Attached Storage* (NAS)

Stockage distant par blocs

- iSCSI (*Internet Small Computer System Interface*)
  - Connexion TCP dans couche de transport SCSI
    - Conçu pour la longue distance
- AoE (*ATA over Ethernet*)
  - Blocs ATA transportés dans paquets Ethernet
    - Uniquement pour les LANs

Divers

Exemple des serveurs parallèles

- Réplication données
- Équilibrage de charge
- Facile en lecture seule
  - Pas de cohérence
  - Serveur WWW
- Difficile en écriture
  - Sauf si cohérence peu importante
    - Mise à jour d'un serveur WWW

Éviter les goulets dans les serveurs parallèles

- Ne pas centraliser
  - Distribuer et synchroniser
- Ou centraliser le minimum
  - Serveur de metadonnées qui distribue la charge de travail à de multiples serveurs
    - Serveurs de fichiers
      - Lustre, PVFS, NFSp, ...

Répartir les serveurs

- Toujours être près des clients
  - Géographiquement
    - Réduction des temps d'accès
- Et s'adapter à leur spécificité
  - Langue, besoins, nombre, ...

Serveur réseau répliqué

- Partage en lecture essentiellement
  - Moteurs de recherche
- Plusieurs processus
  - Recouvrement des entrées-sorties
  - Partager les données entre processus de machines multiples
- Proxy redirige vers un des serveurs physiques
  - Équilibrer la charge
  - Changer de serveur en cas de panne

Serveur réseau répliqué (2)

- Réplication de sockets ?
  - Difficile avec système standard
- Routeur évolué au lieu de proxy
  - Capable de migrer une connexion
- Migrer extrémité socket sur autre serveur ?
- Utile si connexions à longue durée
  - Connexions courtes souvent négligeables
    - Sauf si système critique

IP *Spoofing*

- Transmettre la requête à un des serveurs
  - Ce serveur répond directement à l'émetteur
    - Il *spoofe* l'IP du premier serveur
- Impossible en mode connecté
  - UDP seulement
    - NFS, RPC, ...
- NFSp