

CaRsync

Sincronització remota de fitxers mitjançant
tècniques de memòria cau al servidor

Projecte de Final de Carrera

Enginyeria en Informàtica

Guillem Cantallops Ramis

Dirigit pel Dr. Ricardo Galli

Estructura de la presentació

- Introducció
 - El problema de la sincronització remota de fitxers
 - Objectius
- Antecedents
 - Tècniques locals
 - Tècniques distribuïdes
- Motivacions
 - Problemes amb `rsync`
 - Solucions proposades
- Disseny del CaRsync
 - Fitxers, missatges i processos
 - Dinàmica: seqüència i concurrència
- Implementació del CaRsync
 - (Diversos subsistemes...)
 - Memòria cau al servidor
- Proves i avaluació
- Conclusions

Introducció: El problema de la sincronització remota de fitxers

- Molt freqüent: actualitzar versions antigues amb versions noves
- Mètodes tradicionals
 - **Còpia íntegra** de la nova versió
 - **Fitxers de diferències** entre versions consecutives
- **Algorisme *rsync***
 - Detecta en cada cas diferències i transmet només això
 - Molt intensiu, causa problemes de càrrega als servidors
 - El programa `rsync` és *stateless* (execucions redundants)
 - El nou `carsync` recorda les diferències ja calculades

Introducció: Objectius

- Objectiu principal
 - Sistema de memòria cau al servidor: sense execucions redundants de l'algorisme *rsync*. **Redueix la càrrega.**
- Objectius secundaris
 - **Nova base de codi:** implementa l'algorisme *rsync* amb les millores, d'una manera clara i mantenible.
 - **Protocol més senzill i sistemàtic:** facilita crear eines compatibles.
 - **Rendiment comparable o millor** que el d'*rsync*: amplada de banda, temps dels clients i càrrega al servidor.
- Evitar: generar incompatibilitats o inconsistències innecessàries

Antecedents: Tècniques locals

- `diff` **expressa les diferències** entre dos fitxers de text estructurats en línies.
- `patch` aplica la sortida d'un `diff` sobre la versió antiga i la **converteix en la versió nova** (o viceversa).
- Problemes:
 - Limitat a **fitxers de text**.
 - Fa falta **accés directe** a les dues versions per comparar.
 - Només funciona **entre dues versions concretes**, normalment successives. Actualitzar la versió X a la versió X+Y implica aplicar Y diferències per ordre.

Antecedents: Tècniques distribuïdes (I)

- **Més general:** el client té el fitxer B i vol actualitzar-lo al fitxer A del servidor aprofitant tantes parts de B com sigui possible.
- Cap dels dos sistemes pot fer comparació directa. L'**algorisme *rsync*** s'executa entre tots dos i permet fer la sincronització:
 - El client divideix B en blocs, calcula les seves signatures (*checksum + hash*) i les envia al servidor.
 - El servidor cerca dins A blocs (a totes les posicions possibles) amb la mateixa signatura que algun bloc de B. Va generant instruccions per copiar literalment dades d'A o per aprofitar blocs de B, i les envia al client.
 - El client seguint les instruccions del servidor reconstrueix A a partir de les dades literals d'A més els blocs que pot aprofitar de B.

Antecedents: Tècniques distribuïdes (II)

- **Avantatges** de l'algorisme *rsync*:
 - *Round trip* únic: minimitza l'efecte de la **latència** del canal de comunicacions.
 - Pot consumir menys **amplada de banda** que copiar A, com més coincidències amb B millor.
- Altres consideracions:
 - Obliga al client i al servidor a fer **càlculs**. El servidor no resultaria viable si no fos pel *rolling checksum*.
 - Sincronització de **més d'un fitxer** en general: *pipelining*.
 - Sincronització d'**arbres de directoris**: confecció prèvia de llistes amb algorismes recursius.

Motivacions: Problemes amb `rsync`

- **Cost intrínsec** de l'algorisme *rsync*
 - No és només entrada/sortida, requereix càlculs.
 - Al servidor és pitjor si soporta usuaris concurrents.
- **Inconvenients superables** del programa `rsync`
 - Dispara la **càrrega del servidor**: N peticions anàlogues fan executar N vegades l'algorisme *rsync* (bastaria 1).
 - No és fàcil millorar-lo, està escrit en C **molt desordenat**.
 - **No està documentat** el protocol.

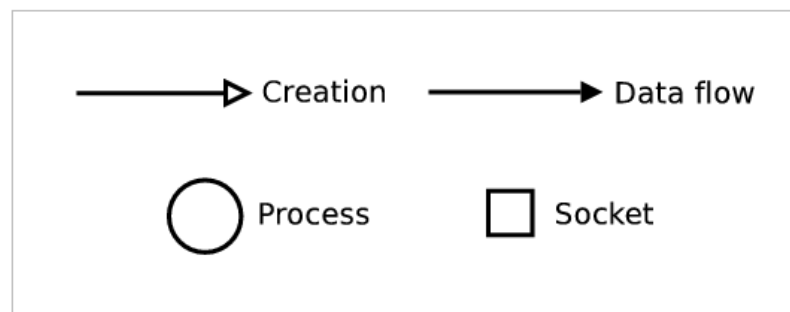
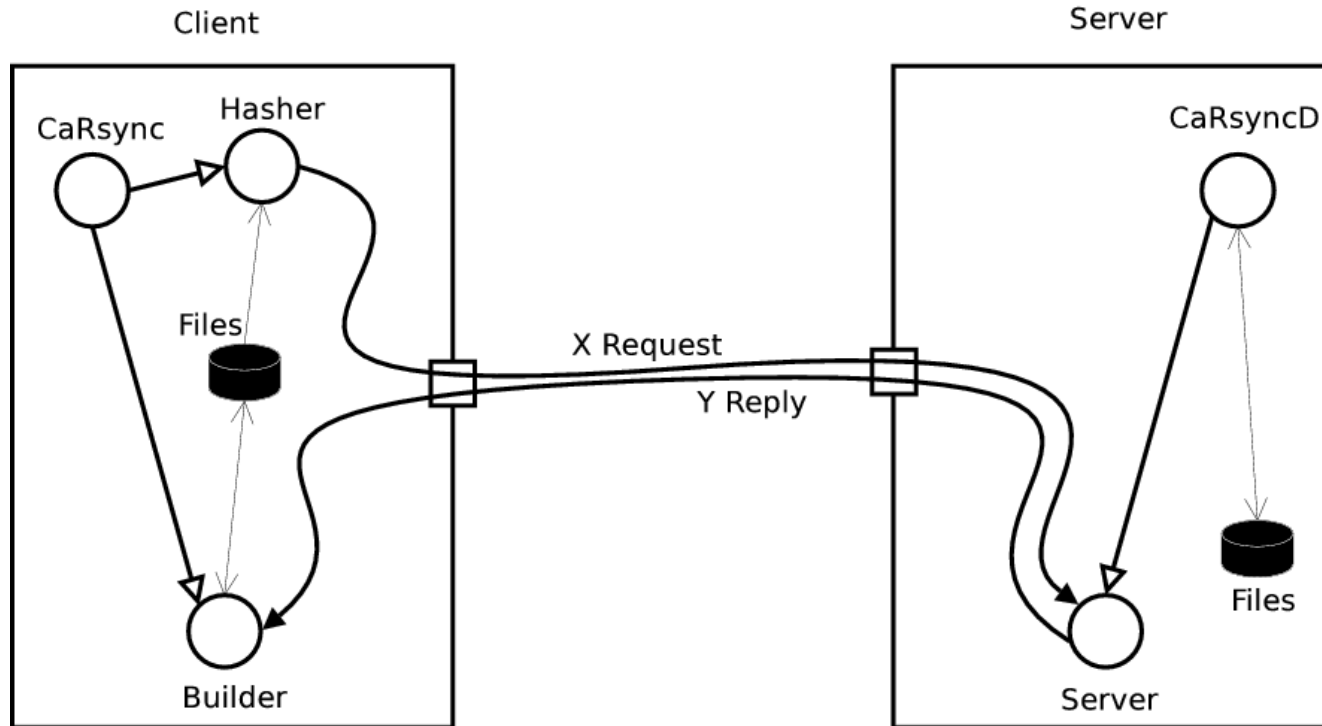
Motivacions: Solucions proposades

- L'algorisme *rsync* i algunes optimitzacions que el fan viable (*rolling checksum*) estan ben explicats pels mateixos autors.
- Per aquest motiu, **es dissenya i s'implementa** el `carsync`
 - Sense tantes opcions com `rsync`, però amb les més usades.
 - Més net i ordenat. Més mantenible i ampliable.
 - Amb un protocol més sistemàtic. Facilita la compatibilitat.
 - Amb un **sistema de memòria cau** al servidor. Recorda les respostes generades i evita recalcular-les per satisfer peticions anàlogues.

Disseny del CaRsync: Fitxers, missatges i processos

- Fitxers
 - A: versió nova
 - B: versió antiga
 - C: còpia d'A
- Missatges
 - X: petició (signatures)
 - Y: resposta (instruccions)
 - Z: salutació
- Processos
 - CaRsync
 - Hasher ($B \rightarrow X$)
 - Builder ($B, Y \rightarrow C$)
 - CaRsyncD
 - Server ($A, X \rightarrow Y$)

Disseny del CaRsync: Dinàmica: seqüència i concurrència



Implementació del CaRsync: Processos

- **CaRsync:** Inicialització, connexió i negociació amb el servidor. Inicia *Hasher* i *Builder* (al client).
- **CaRsyncD:** Inicia *Server* (al servidor).
- **Hasher:** Calcula les signatures dels fitxers B, genera les peticions X i les envia al *Server*.
- **Server:** Reb peticions X d'un *Hasher*, i amb els fitxers A genera les respostes Y per al *Builder*. Nucli de l'algorisme *rsync*: *sliding window*.
- **Builder:** Reb respostes Y del *Server*. Dues comandes bàsiques que executa per ordre per obtenir C:
 - Ya: utilitzar dades literals d'A (dades adjuntes)
 - Yb: utilitzar bloc de B (número de bloc adjunt)

Implementació del CaRsync: Connexions

- Socket **TCP**
 - Insegur, servidor acceptant clients concurrents.
 - Servidor: `socket()` `bind()` `listen()` `accept()`
 - Client: `socket()` `connect()`
- Sessió **SSH**
 - Segur, compressió, servidor 1 cop a petició del client.
 - Client: `socketpair()` `fork()` `dup2()` `execlp()`
 - Servidor: un sol procés *Server*.
- En tot cas el resultat sempre és **descriptor de fitxer**.

Implementació del CaRsync: Taules de dispersió

- Necessàries per al **sistema de signatures** del procés *Server*.
- Ús que en fa el *Server* quan executa l'algorisme *rsync* :
 - Les signatures rebudes (petició X) s'insereixen a una T.D.
 - Per a cada bloc d'A (a totes les posicions) el *rolling checksum* es cerca dins les signatures dels blocs de B.
 - Si coincideix amb algun, es calcula el *hash* del bloc d'A i es compara amb el del bloc de B per confirmar la coincidència.
- Funcionament:
 - *Adreçament obert*
 - *Doble dispersió*

Implementació del CaRsync: Recorregut recursiu d'arbres

- Algorisme de sincronització a un nivell més alt: determina quins fitxers individuals s'han de...
 - **Sincronitzar** (amb l'algorisme *rsync*)
 - **Copiar** (íntegrament)
 - **Esborrar**
- Client i servidor construeixen la llista ordenada del que tenen. De la *comparació implícita* s'extreuen les conclusions.
- S'amplia un TAD “llista” típic per gestionar llistes de fitxers (nom i altres metadades) obtingudes del recorregut recursiu d'arbres de directoris.

Implementació del CaRsync: Funcions de dispersió

- Funcions **no invertibles**. Una entrada de qualsevol longitud produeixen una sortida de longitud fixa. Tot canvi a l'entrada provoca grans canvis a la sortida.
- L'algorisme *rsync* es basa en l'aplicació de dues funcions d'aquest tipus sobre cada bloc:
 - **Checksum**: *A32*. Dissenyat originalment per a l'algorisme *rsync*. És un *rolling checksum*.
 - **Hash**: *MD5*. Molt robust però també molt costós, i s'ha de calcular íntegrament per a cada bloc.

Implementació del CaRsync: Tamany dels blocs. Arguments.

- El **tamany dels blocs** influeix en l'efectivitat de l'algorisme *rsync*.
 - Grans: poques signatures però també poques coincidències.
 - Petits: moltes signatures però també moltes coincidències.
- Heurística recomanada:
 - Tamany del Bloc ~ $\text{SQRT}(\text{Tamany del Fitxer})$
 - Signatures ~ 1% del T.F.
- Els **arguments** (minúscula activa, majúscula desactiva) són:
 - c: Sistema de memòria cau.
 - d: Esborrat de fitxers desapareguts.
 - p: Conservació de metadades.
 - s: Connexió segura via SSH.

Implementació del CaRsync: Gestió d'errors. Desenvolupament.

- **Errors** sempre per `stderr`.
- Sistema senzill de pseudo-excepcions amb condicionals i macros.
- **Error d'exemple:** `PID=24269 carsyncd.c[49]: main(): Binding master socket: Address already in use`
- **Desenvolupament** amb Programari Lliure.
- Sistemes de control de versions CVS i SVN.
- Eines de programació: `vim`, `gcc`, `make`, `bash`, `perl`.
- Estructura típica de codi C en fitxers `*.c` i `*.h`.
- Documentació: LyX, LaTeX (teTeX), Dia, OpenOffice.org.

Implementació del CaRsync: Memòria cau al servidor

- Principal innovació del `carsync`: recorda execucions passades de l'algorisme `rsync` i evita repetir-les sobre les mateixes dades
- **Entrades** del sistema de memòria cau: fitxers que contenen respostes Y. La clau (basada en peticions X) és el nom.
 - Lectura: S'utilitza la clau per cercar. En cas de *hit* ja no s'executa l'algorisme `rsync`: s'obté la resposta del cau.
 - Escriptura: En cas de *miss* s'executa l'algorisme `rsync` i a més d'enviar la resposta Y al client, es guarda al cau.
- **Concurrència**: `flock()` amb `LOCK_SH+LOCK_NB` (lectura), `LOCK_EX+LOCK_NB` (escriptura) i `LOCK_UN` (desbloqueig).

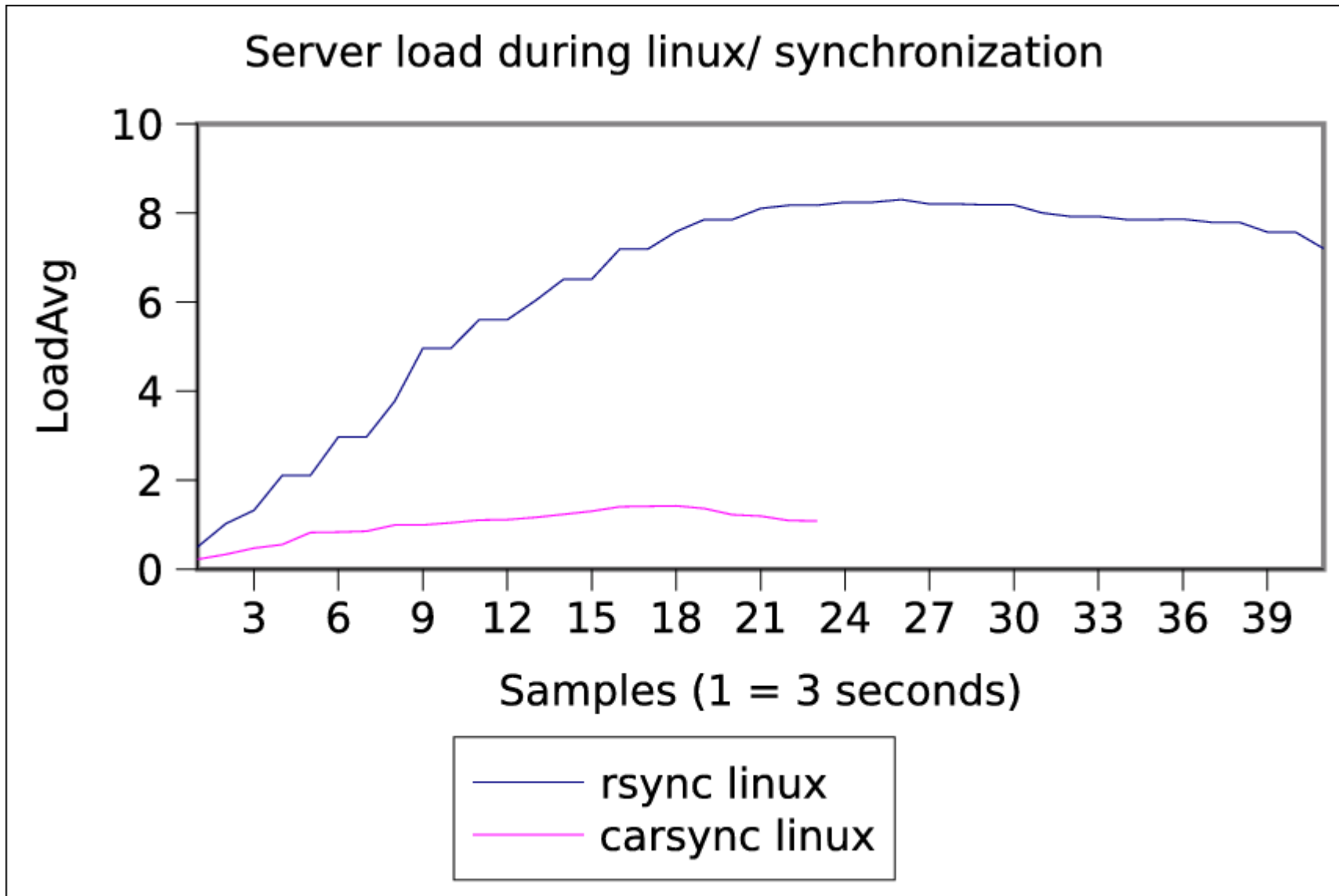
Proves i avaluació (I)

- Fitxers de prova: `vmlinux`, `packages`, `linux.tar`, `linux/`
- Clients: proves amb 1 i 10 clients.
- Proves relativament sintètiques (realitat = combinació).
- **Amplada de banda** (dades enviades i rebudes)
 - Similars en general. El `carsync` és *lleugerament* millor o pitjor segons el cas.
 - Amb un sol fitxer gran `carsync` envia 400KB en lloc de 100KB, però reb 177MB en lloc de 208MB per fer *la mateixa sincronització*.
 - Amb molts de fitxers petits `carsync` envia uns 5MB en lloc de 2MB, i reb 77MB en lloc de 65MB.

Proves i avaluació (II)

- **Temps i càrrega** (execució dels clients, càrrega del servidor)
 - El `carsync` sense cau tarda un temps (a cada client) comparable a l'`rsync`. Amb el cau activat i no poblat tarda més ja que l'ha de poblar. Amb el cau activat i poblat és sensiblement més ràpid.
 - Amb clients concurrents, el `carsync` sense cau o amb cau no poblat provoca càrregues (al servidor) comparables a les que provoca `rsync` (N peticions : N execucions *rsync*).
 - Amb cau poblat `carsync` provoca càrregues molt més baixes, comparables al cas d'un sol client (escala *quasi constant*, N:1) i `rsync` escala de manera *quasi lineal* (N:N).

Proves i avaluació (III)



Conclusions

- **Objectius assolits**

- Amb el sistema de memòria cau, la càrrega del servidor quan clients concurrents fan peticions anàlogues és molt més baixa, tendeix a $O(1)$ vs. $O(n)$ inicial.
- Per disseny, els programes i el protocol són més clars, senzills, i reutilitzables.
- Rendiment de la nova implementació de l'algorisme *rsync*: *comparable* al de la del programa `rsync`.

- **Futur del projecte**

- Possibles millores tècniques: heurística del tamany dels blocs, reducció de les signatures i compressió.
- Programari lliure sota llicència GNU GPL. La documentació i el desenvolupament continuen.