SEPARATING DATA AND CONTROL TRANSFER IN DISTRIBUTED OS

STUDENTE: ANTONIO ALLOCCA

DOCENTE: GIUSEPPE CATTANEO

CONTESTO STORICO

I miglioramenti portati alle architetture degli elaboratori hanno portato a workstation con la potenza di calcolo di 1000+MIPS.

I miglioramenti ci sono stati anche nell'ambito delle telecomunicazioni con **reti sempre più efficienti ed affidabili.**

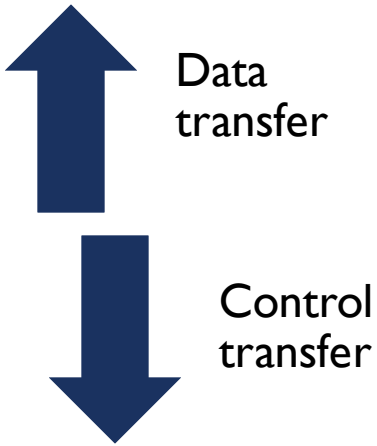
Ciò ha dato la possibilità di intravedere un futuro in cui ci sia un accoppiamento sempre più stretto nei **sistemi hardware distribuiti a livello hardware e il software** si deve adattare di conseguenza.

IDEA PRINCIPALE

In questo paper viene proposto un modo alternativo per **strutturare un sistema distribuito** in modo da sfruttare il modello di comunicazione basato **sull'accesso remoto alla rete** a **segmenti di memoria protetti**.

key future

Separazione di:



Activity	Number of Calls	Percentage of Total
Get File Attribute	8960671	31
Lookup File Name	8840866	31
Read File Data	4478036	16
Null Ping Call	3602730	12
Read Symbolic Link	1628256	6
Read Directory Contents	981345	3
Read File System Stats.	149142	0.5
Write File Data	109712	0.4
Other	109986	0.3
Total	28860744	100

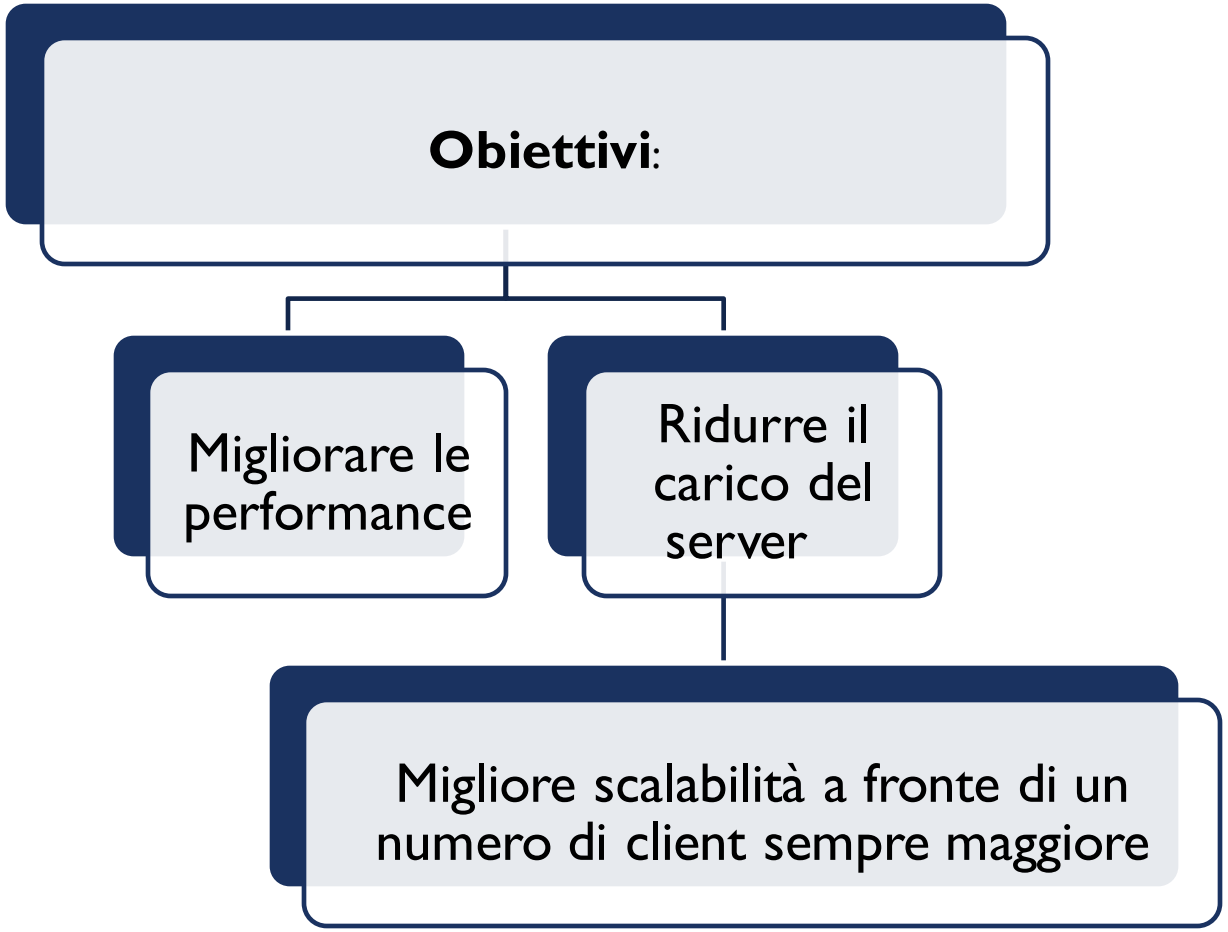
Table 1a: Summary of NFS RPC Activity

Activity	Network Traffic (Mb)		Control / Data
	Control	Data	
Get File Attribute	214	481	0.44
Lookup File Name	246	761	0.32
Read File Data	185	2176	0.09
Read Symbolic Link	59	19	3.10
Read Directory Contents	54	1862	0.03
Read File System Stats.	4	3	1.33
Write File Data	4	271	0.01
Overall Total	766	5573	0.14

Table 1b: Breakdown of NFS RPC Traffic

OBIETTIVI

Tale caratteristica è in contrasto con la struttura classica dei sistemi distribuiti che sono organizzati tramite scambio di messaggi oppure RPC.



OPERAZIONI CONCESSE

Il modello di comunicazione proposto fornisce un **insieme di primitive** al fine di accedere tramite processi a una **memoria remota contigua virtuale** di un altro processo.

Operazioni concesse ai processi quando si accede alla memoria (Bisogna avere i permessi per farlo):	Read
	Write
	Compare-And-Swap (**CAS**)

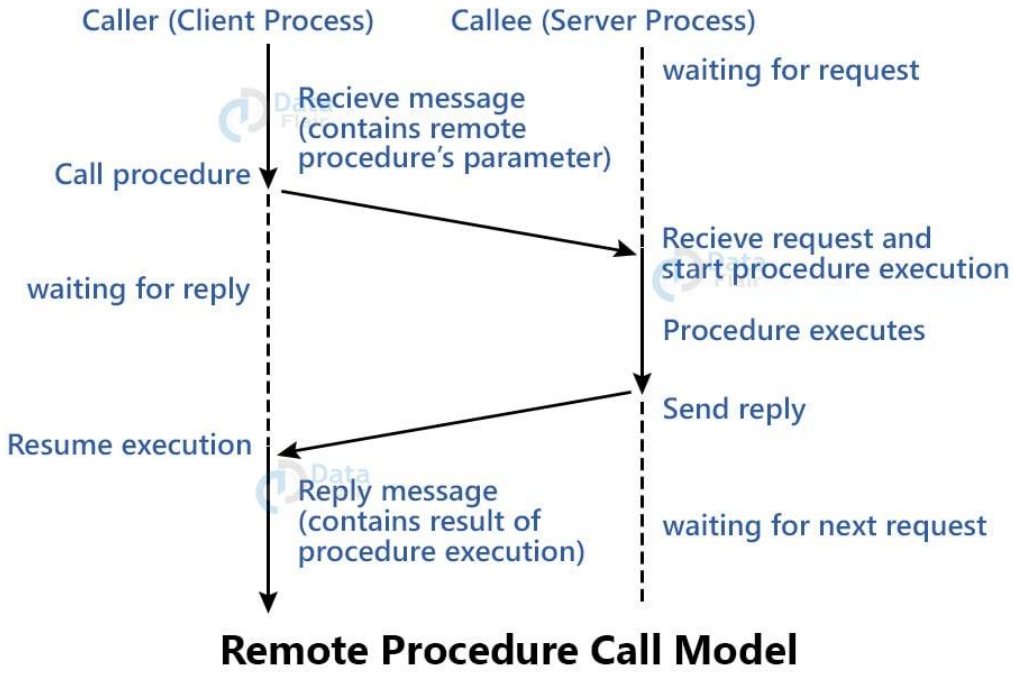
RPC

RPC era allora il **meccanismo di comunicazione** predominante tra le componenti di un **sistema distribuito**, esso operava sostanzialmente in due semplici funzioni:

1. Trasferimento dei dati tre lo spazio di indirizzamento del client e quello del server. Tale implementazione può essere costosa per via di alcuni parametri come:
    1. Il client può richiedere di effettuare il marshalling/unmarshalling dei parametri per la trasmissione
    2. La copia dei dati tra il clien o il server e la rete. (protocollo SMB)
2. Trasferimento del controllo da un thread del client a un thread del server e viceversa. Per effettuare il controllo del trasferimento implica diverse azioni:
    1. Bloccare il thread del client che porta al rescheduling del processore del client
    2. Elaborare il pacchetto RPC nell'OS di destinazione
    3. Schedualre, assegnare ed eseguire il thread del server
    4. Reschedulare il processore del server non appena il pacchetto viene restituito
    5. Elaborazione del pacchetto di risposta da parte dell'OS del client
    6. Schedulare e riprendere il thread originale del client

PROBLEMA DI RPC

Quindi per copiare anche un solo byte dal client al server, bisogna effettuare sia il trasferimento che il controllo che neanche i sistemi più ottimizzati riescono ad ottenere riduzioni sostanziali dei tempi.



Remote Procedure Call Model

COMPONENTI DEL SISTEMA DISTRIBUITO

Viene esaminata l'applicazione NFS file server per un gruppo di **80-100 workstations** con **disco locale**. Il file server fornisce:

1. X-terminal fonts
    1. https://en.wikipedia.org/wiki/X_terminal
    2. https://en.wikipedia.org/wiki/X_Window_System
2. Source tree (Directory che contiene il source del kernel)
3. /usr che contiene i binari degli eseguibili.

NUOVA STRUTTURA PER LE APPLICAZIONI DISTRIBUITE

L'obiettivo è migliorare le **performance e la scalabilità** dei servizi distribuiti e per fare ciò ci sono 3 **obiettivi di design**:

- Il sistema di comunicazione deve raggirare il **bottleneck** che può verificarsi quando viene richiesto un servizio.
- **Riduzione del carico** da parte del **server** per fornire scalabilità
  - Affidando più carico ai client
- **Riduzione del carico sulla rete**

COMUNICAZIONE NELLA LAN

L'obiettivo è di sviluppare un **cluster di workstation collegati in LAN** e nello studio non viene calcolato il **data loss** in quanto viene classificato come un evento molto raro per i controlli di flussi fatti all'interno della LAN stessa.

Tale scelta permette l'uso di **primitive di comunicazione più semplificate**, in particolare viene usato un modello di comunicazione basato sulle nozioni del **remote network memory**.

In tale modello si può leggere e scrivere direttamente all'interno dello spazio di indirizzamento virtuale dei processi che sono presenti su altri computer all'interno della rete

- **Data trasfer only**
  - Non è richiesta nessuna cooperazione con il processo remoto nè in lettura e nè in scrittura all'interno del suo spazio di indirizzamento

MODELLO DI COMUNICAZIONE

Abbiamo un **insieme di primitive di comunicazione** per accedere ad una memoria remota

A livello astratto: Insieme di **segmenti di memoria** e operazioni **(read,write)** definite su di esse

Le **operazioni** sui dati sono supportate attraverso «meta-istruzioni»

Le applicazioni **scambiano** tali segmenti tramite un protocollo a più alto livello implementato da un segmento che prende il nome di **«server»**

I segmenti sono **protetti** da accessi non autorizzati tramite **grant/revoke** dei permessi

SEGMENTI

Segmenti = Pezzi contigui della memoria virtuale dell'utente

- Definiti da applicazioni a livello utente
- Controllati tramite descrittori in possesso a controller e software privilegiato.

ISTRUZIONI DI MEMORIA

Il modello di memoria remota è definito come un insieme di meta-istruzioni su coprocessore

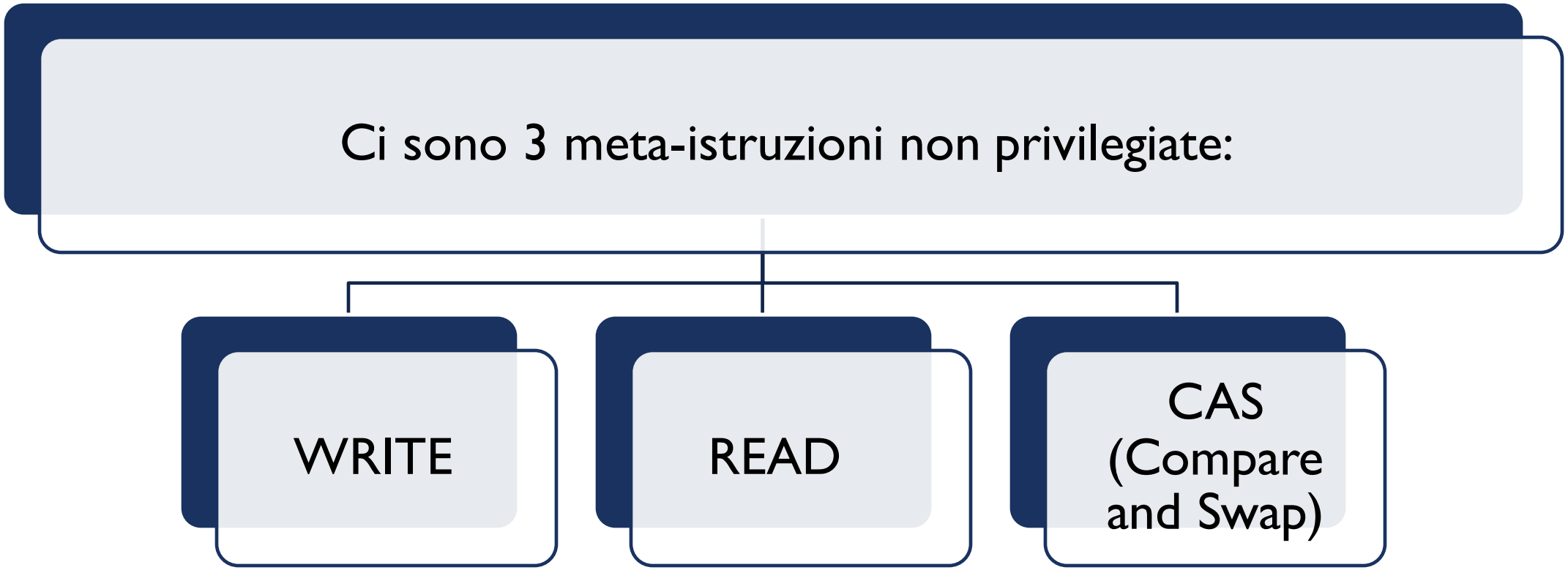
(Il fatto che le istruzioni siano definite su co-processore possono essere efficientemente emulate senza richiedere un hardware apposito)

Il coprocessore contiene descrittori che definiscono segmenti di memoria remota.

Ogni descrittore contiene:

- Dimensione del segmento
- Indirizzo del nodo remoto
- Informazioni di protezione

META-ISTRUZIONI



WRITE (STRUTTURA)

RD	• Registro del descrittore all'interno del coprocessore che identifica un segmento di memoria remoto
OFF	• Offset dal byte iniziale nel segmento per la WRITE
COUNT	• Numero di byte che devono essere scritti
NOTIFY	• Indica se la destinazione remota (tipo un server) deve essere notificata quando i dati la raggiungono

Quando una write viene eseguita, il **coprocessore verifica i permessi** di chi vuole accedere alla memoria. Se il controllo va a buon fine allora il pacchetto viene spedito.

ULTERIORI CARATTERISTICHE DELLA WRITE

Quando una write viene eseguita, il coprocessore verifica i permessi di chi vuole accedere alla memoria. Se il controllo va a buon fine allora il pacchetto viene spedito.

Operazione non bloccante

Quando una write viene completata localmente, il coprocessore garantisce che il dato è stato accettato dalla rete, non che è arrivato a destinazione.

Quando viene ricevuta una richiesta di write, il co-processore remoto utilizza [rd,off,count] per validare la richiesta

Il descrittore identifica uno spazio di indirizzamento virtuale all'interno di un processo

Il coprocessore legge le address translation tables per quel processo e legge i dati in memoria

READ (STRUTTURA)

RS + SOFF	• Identificano il segmento remoto e l'offset dove i dati da leggere possono essere trovati
RD + DOFF	• Identificano il segmento locale e l'offset dove i dati sono depositati

ULTERIORI CARATTERISTICHE DELLA READ

1) La richiesta non è bloccante
2) Il dato restituito dal processore remoto è messo nello spazio di indirizzamento di chi fa la richiesta
3) Non c'è bisogno di specificare di message registers (https://blog.wildix.com/understanding-register-method/) semplificando le istruzioni
4) Notify indica se chi legge deve essere accertato quando la read ritorna i dati. Senza di esso, chi invoca la read non ha idea di quando i dati vengano ricevuti. (L'unico modo di capire quando arriva il messaggio è di controllare ripetutamente la celle di memoria in cui il dato dovrebbe trovarsi.)

MODELLO A MEMORIA REMOTA > MESSAGE PASSING PER IL TRASFERIMENTO DI DATI... PERCHÉ?

Message passing

- Specifica solo gli end-point della comunicazione (socket) il che fa aumentare l'overhead durante il demultiplexing e la copia dei dati.

Modello a memoria remota

- I pacchetti e i dati immagazzinati specificano la destinazione del dato in memoria

CAS (STRUTTURA)

RS	SOFF	OLD-VALUE	RD	DOFF	NEW-VALUE

Viene effettuato un confronto sul dato remoto finché **old-value** non è trovato, a quel punto viene sostituito con **new-value**.
Il risultato è 0 o 1.

FUNZIONI AGGIUNTIVE

Necessitiamo di altre funzioni più specializzate per garantire le seguenti caratteristiche (read e write non bastano a soddisfare tutte le caratteristiche):

1) Descriptor maintenance
2) Import ed Export di segmenti
3) Pinning e Unpinning di pagine di memoria virtuale (Application-based)
4) Permettere la **sincronizzazione**
5) **Controllo del trasferimento**

NOTIFY

Il **controllo del trasferimento** è separato dall'invio dei dati, questo **può essere compensato** attraverso il campo **notify** ad esempio.

Ogni descrittore del segmento ha un campo "**Notify**" che può avere 3 stati:

1) **Always** notify
2) **Never** notify
3) **Conditionally** notify (Dipende dal **campo notify** del pacchetto)

Il **trasferimento** di dati **non è legato** all'esecuzioni di **thread o procedure**.

LA STRUTTURA DELLE APPLICAZIONI

La struttura ha 4 principali componenti:

1. Client e Server
2. Meccanismi di trasferimento dati specializzato (accesso alla memoria remota in modo diretto e protetto)
3. Server Clerks eseguiti sui client
4. Clerk-to-Server Data Transfer

In alcune implementazione possiamo avere anche la rimozione totale del server, lasciando che i clerk mantengano lo stato de soli.

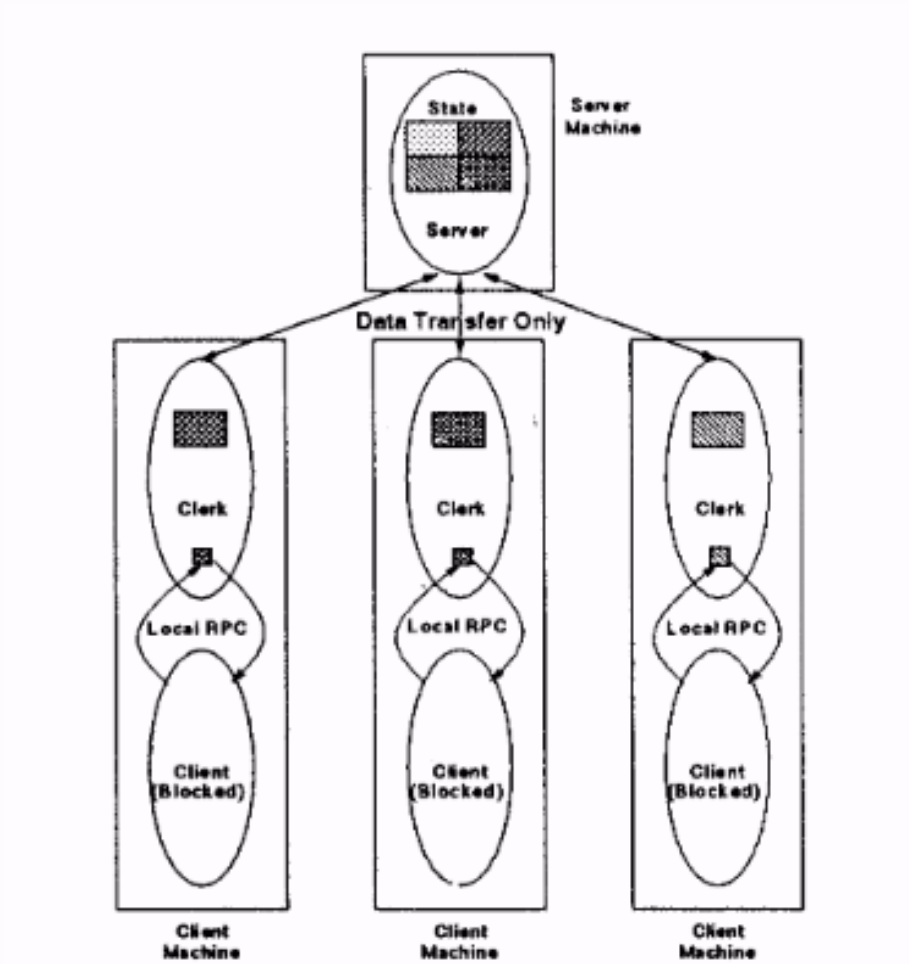


Figure 1: Structuring a Distributed Service Without RPC

BENEFICI DELLA STRUTTURA

Tale struttura permette di ottenere delle caratteristiche uniche:

1) **Local cashing** per ridurre la comunicazione tra le macchine
2) Viene **eliminato il controllo del trasferimento** dei dati tra le macchine (grazie ai clerk si può ottenere solo il trasferimento dei dati)
3) Il trasferimento bidirezionale dei dati è **ottimizzata** (utilizzo dei clerk)

L'utilizzo dei clerk fornisce un livello di **astrazione** simile a quella fornita dalle RPC.

PROBLEMI DELLA STRUTTURA

Nel sistema lo **stato del server** è distribuito e tenuto in una **cache locale**. Ciò porta ad avere **problemi di chache coerence** e **consistenza** che devono essere risolti se il client ne fa esplicita richiesta.

Non viene esplicitamente richiesto che il **broadcast** sia supportato dalla rete. Infatti gli autori dell'articolo non pensano che esista una singola policy per la cache coerence adattabile a tutti i servizi. La struttura proposta permette al singolo servizio l'uso di schemi più convenienti per il servizio stesso.

LOCALIZZARE DATI REMOTI

Un'altra importante caratteristica è che i sistemi che sono distribuiti fanno parte della stessa applicazione.

Ciò permette **un'ulteriore ottimizzazione** in quanto non c'è bisogno si scambi di messaggi, ma il clerk del server è a conoscenza di tutte le strutture dati.

SINCRONIZZAZIONE

La struttura proposta gestisce la sincronizzazione con i metodi più comuni all'interno dei sistemi distribuiti, in particolare abbiamo 4 modalità in cui la gestione viene gestita

- Evitare la sincronizzazione quando possibile, anche in presenza di variabili condivise
- Sfruttare le proprietà di atomicità fornite dal modello di comunicazione
- Utilizzare operazioni CAS per creare primitive di sincronizzazione
- Implementando il controllo del flusso rendendolo simile al metodo con RPC

SICUREZZA

I client accedono ai Clerk tramite RPC locali i quali fungono anche da **firewall**. Per questo i client non possono danneggiare né il server né i loro stessi clerk.

Nel sistema così progettato **c'è fiducia verso tutti i componenti** ma come in un NFS potrebbe essere richiesta l'implementazione di **tecniche crittografiche**. Ciò porta a dover **cifrare e decifrare** tutti i dati inviati tramite write e tutti i dati letti tramite read. Se la cifratura e la decifratura viene simulata con software, le prestazioni calano di molto per cui gli autori propongono **soluzioni hardware**.

CACHE CONSISTENCY E RECOVERABILITY

La struttura proposta fa uso del **local cashing** dei dati e dello stato, quindi i meccanismi di **cacache consistency e recupero** a seguito di un crash sono molto importanti.

Però solo con le primite di read e write non si può ottenere un meccanismo efficiente di **fault-tollerance**.
Questo porta ad utilizzare meccanismi di **timeout** che sono semplici da implementare e anche leggeri.

ESEMPI

Semplice Name Server

File Service Distribuito

SEMPLICE NAME SERVER

Scopo del name server: Mantenere il registro di nome e informazioni dei segmenti affinchè chi li importa e chi li esporta può comunicare.

Il name server è organizzato come un insieme distribuito di clerk (uno per macchina) **senza un server centrale**.

Il name server implementa 3 procedure:

1) **Aggiungere** informazioni sui nomi dei segmenti esportati
2) **Cercare** informazioni sui nomi
3) **Cancellare** nomi

I **client** del name server non sono utenti ma sono i **kernel**.

I clerk invece avviano periodicamente dei **refresh della cache dei nomi importanti**. Questo viene fatto per fornire, a seguito di un'operazione di **lookup**, i dati sempre aggiornati e le entries che non sono più valide vengono cancellate dalle **tabelle del kernel**.

IMPLEMENTAZIONE NAME SERVER

In clerk sono creati all'avvio del sistema ed esportano **segmenti well-known** su cui è garantito il privilegio di scrittura. Successivamente vengono **importati** i segmenti well-known dalle altre macchine su cui in futuro si vorranno fare operazione di **lookup**.

Ogni segmento well-known funge da **registro** per immagazzinare informazioni sugli altri segmenti ed è organizzato come un **open-addressed hash table**.

FILE SERVICE DISTRIBUITO

I file system distribuiti come un **NFS** sono l'esempio più comune di **servizio distribuito**.
Naturalmente molti file system distribuiti **usano client e server basati su RPC**.
La maggior parte del traffico in un file system ha bisogno di coinvolgere solo i traffico dei dati.
Rimuovendo però il controllo del flusso presente nell' RPC, si ottengono **2 importanti vantaggi**:

1) Abbassato l'overhead di context switching, invocazioni di procedure ed eventuali blocchi
2) Eliminazione dell'overhead dato dall'elaborazione di dati non necessari

IL MODELLO

Ogni client esegue un server clerk.
Sia il clerk che il server immagazzinano dati all'interno della cache. Questo perché anche i tradizionali NFS si comportano allo stesso modo, solo che il kernel del client funge da clerk.

Siccome vengono immagazzinati dati all'interno della cache, allora avere delle policy di cache-consistency diventa fondamentale.

LE CACHE

Le cache vengono organizzate in aree distinte che contengono ognuna un'informazione specifica. Questo migliora il trasporto di informazioni in quanto i server clerk lato client possono cercare le strutture dati richieste dal server e trasferirle senza dover adottare il controllo del trasferimento.

Le aree:

- File data
- Name Lookup Data
- File attributes
- Directory Entries

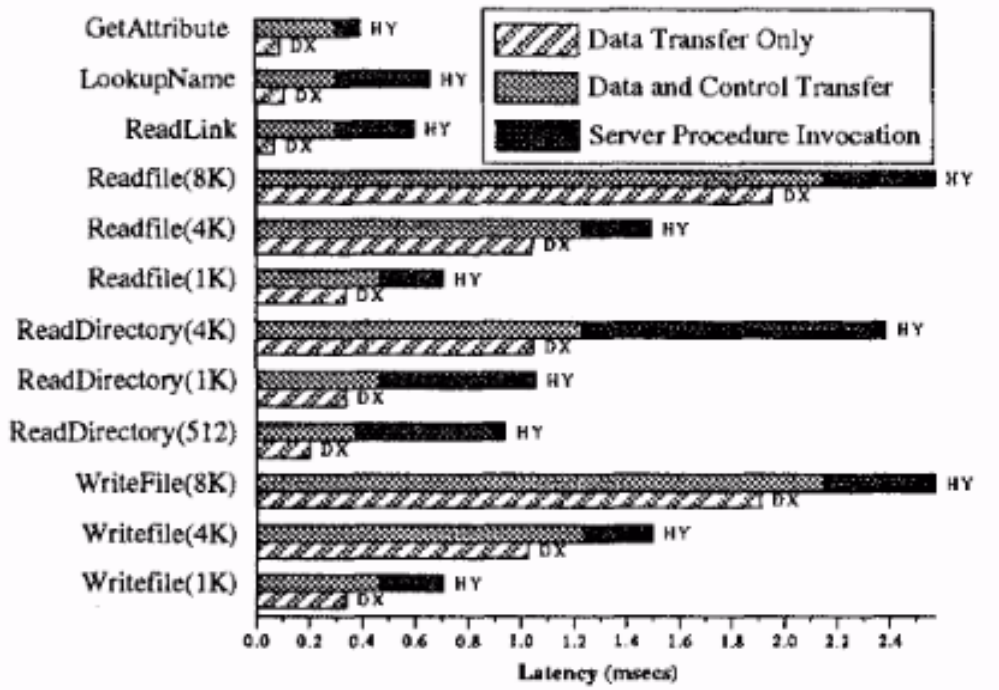


Figure 2: Request Processing Latency Seen by Client

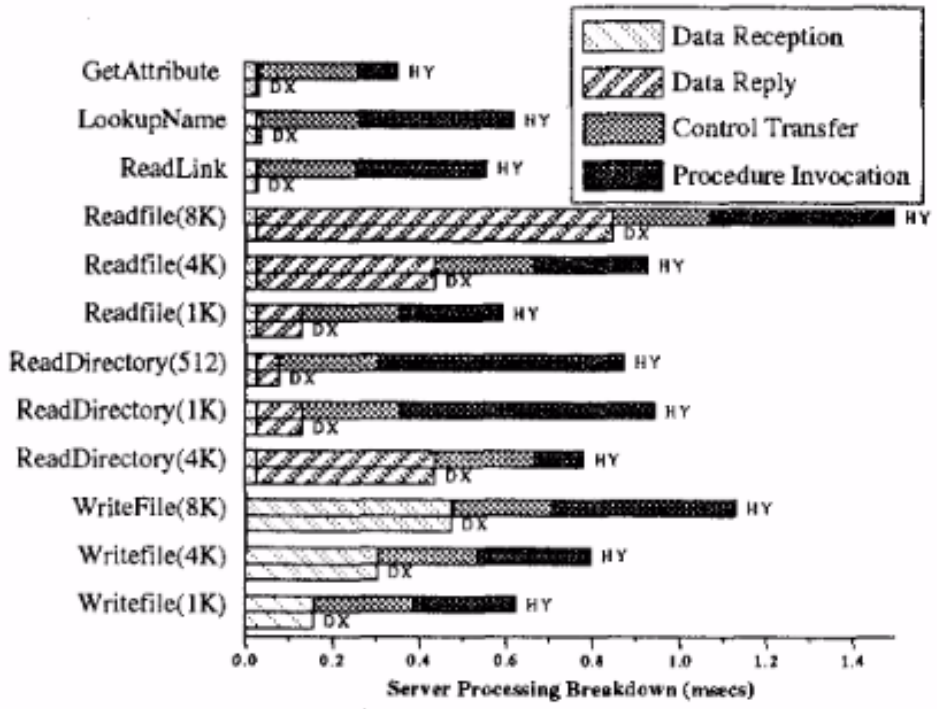


Figure 3: Breakdown of Server Activity

“I cambiamenti della tecnologia porteranno a ripensare la struttura dei sistemi distribuiti. „

CONCLUSIONI

Il paper illustra una nuova struttura per sistemi distribuiti.
Lo scopo è quello di svecchiare il modello basato su RPC per alcuni motivi principali

- Paradigma di programmazione semplice (pure poche primitive)
- La rete ha poca banda e molta latenza
- L'accessibilità di una risorsa non è predicibile

GRAZIE PER L'ATTENZIONE