MANAGEMENT DEI PROCESSI

PROCESSI *entità di esecuzione nel sistema*

Processo sequenziale
come esecuzione sequenziale di azioni sui dati

Le **risorse** usate dal processo sono fornite
o alla creazione del processo
o su richiesta dello stesso

Sono necessari costrutti o primitive
per *definizione dei processi*
creazione/distruzione dei processi
acquisizione/rilascio delle risorse

Meccanismi per i processi
descrittori dei processi
stati dei processi

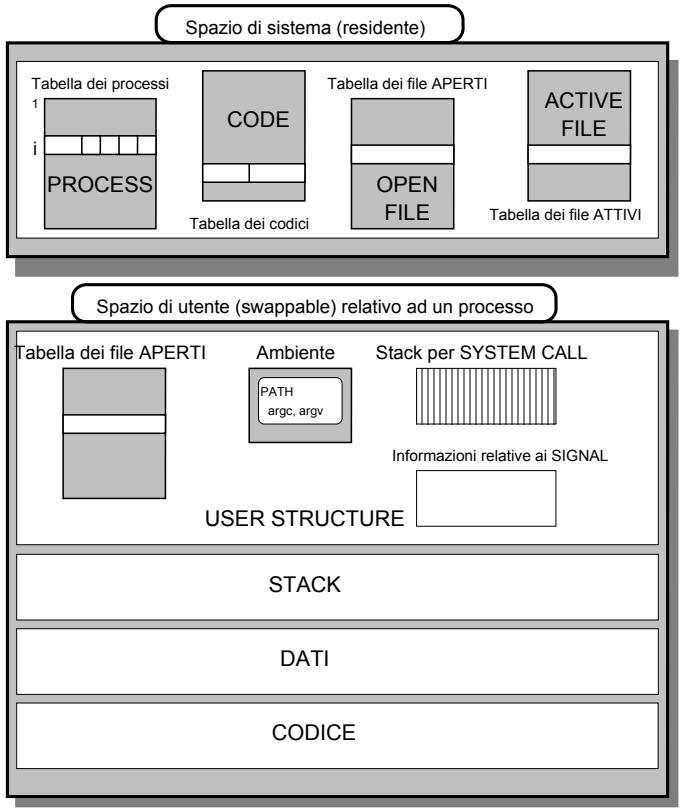
Nel distribuito

Problema fondamentale
Allocazione dei processi ai processori
(statica e dinamica)

modello dei processi

processi pesanti/ processi leggeri

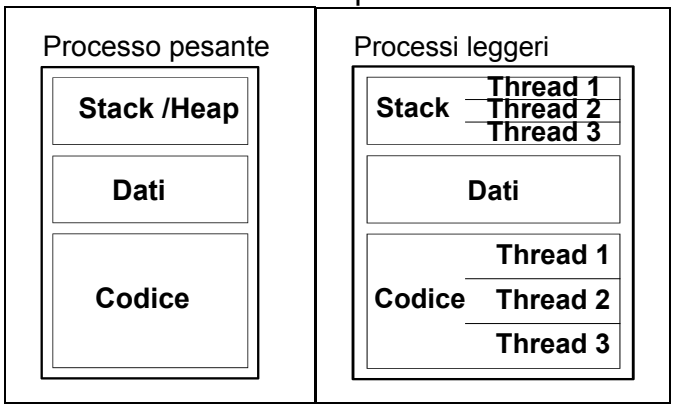
Unix



ad esempio anche, i **thread** o **l**ight**w**eight **p**rocesses

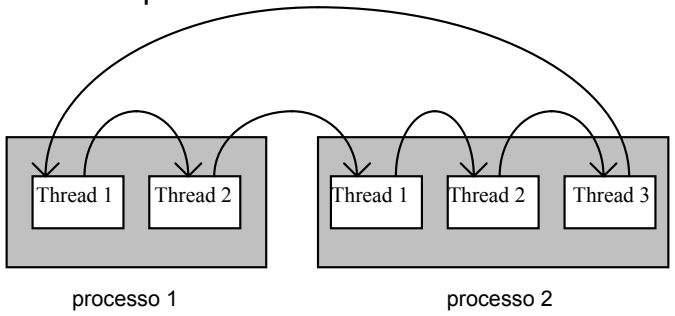
Processi

Processi pesanti:
costo elevato dello scheduling
Processi leggeri e medi:
riduzione del costo
più facile condivisione e comunicazione
Condivisione delle risorse tra processi utente



UNIX, Charlotte solo processi pesanti
V-Kernel processi pesanti e medi
Mach processi leggeri

Lo **scheduler** quindi deve tenere conto dei **thread**

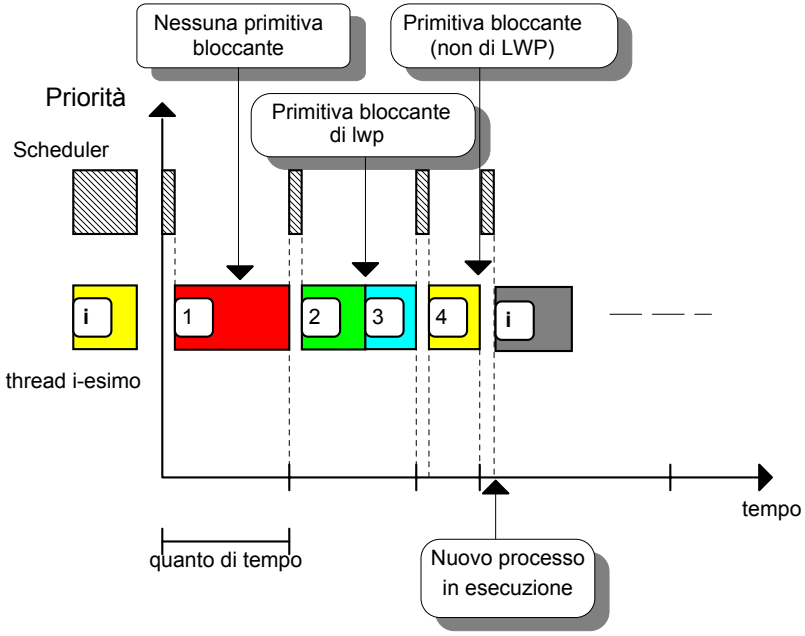


In caso di sistemi esistenti

La realizzazione dei processi leggeri senza modifica del kernel

Problema delle **primitive sospensive**
Se un thread esegue una primitiva sospensiva, viene sospeso l'intero processo (anche tutti gli altri thread)

==>
Si introducono soluzioni ad-hoc



Soluzioni

1) Uso di **primitive ad-hoc**, che non interessano il kernel
2) Uso di primitive non sospensive solamente
   select e **primitive asincrone**
   In un **wrapper** che avvolge la chiamata alla primitiva e garantisce la non sospensione

lwp **L**ight**W**eight **P**rocesses (per SUN BSD)
libreria al disopra del kernel

Gestione dei thread LWP

Inizializzazione

```
int pod_setmaxpri(maxpri)
  int maxpri;
```

Il main è il thread con la massima priorità disponibile e prosegue in concorrenza a thread creati

```
lwp_create(tid,fun,prio,flag,stack,
                  nargs,arg1,.,argn)
   thread_t *tid;
   void (*fun)();   /* codice */
   int      prio; /* priorità thread */
   int      flag;
   stkalign *stack;
   int      nargs;
   int      arg1,...,argn;
```


scheduling dei thread
Un **thread** esegue come dispatcher degli altri
e realizza una **politica di scheduling**

***round-robin*:**

```
      while(1)
    { lwp_sleep(TIME_SLICE);
      lwp_resched(PRIORITA);
    }
```

anche altri schemi
un thread generato per ogni richiesta ed un **dispatcher implicito**
a **pipeline**

I package devono fornire anche
strumenti di sincronizzazione tra **thread**

ad esempio strumenti ad hoc distinti da quelli di kernel
semafori di mutua esclusione
variabili condizione
sospensioni temporizzate

NOTA

In Java, i thread che condividono la macchina virtuale non possono fare azioni di kernel senza tenere conto degli altri
una chiusura di socket la chiude per tutti
essendo mappata nella primitiva di kernel

DCE Distributed **C**omputing **E**nvironment

definito a livello di Open System Foundation
OSF

package standard per fornire i thread
disponibile in System V

Funzioni di:
- creazione/distruzione di thread
- creazione di template per i thread
- gestione dei costrutti di mutua esclusione
- gestione delle variabili condizione
- creazione e gestione di variabili condivise dai thread
- gestione delle autorizzazioni

- altri servizi: autenticazione, gestione risorse

Non sono necessarie funzioni di **kernel** ad-hoc
per le **primitive**

Il sistema DCE rappresenta un accordo tra
realtà di mercato proprietario diverso e
standard accademici accettati

RPC integrate con i processi leggeri: ogni operazione viene eseguita in un **thread** generato

Soluzioni in kernel vs. applicative

User-level
kernel level

☺ efficienza rispetto a processi pesanti

gestioni nel kernel: kernel-level thread (Mach, V)

☺ integrazione degli strumenti nel kernel

☹ meno efficienti di realizzazioni a livello applicativo
politiche general-purpose

gestioni applicative: User-level thread

☺ possibilità di variare le politiche secondo necessità

☹ meno efficienti di realizzazioni ad hoc
interferenza con le politiche di kernel

gestioni miste: FastThread

☺ possibilità di integrare le politiche secondo necessità
e
di ottenere i vantaggi di entrambe

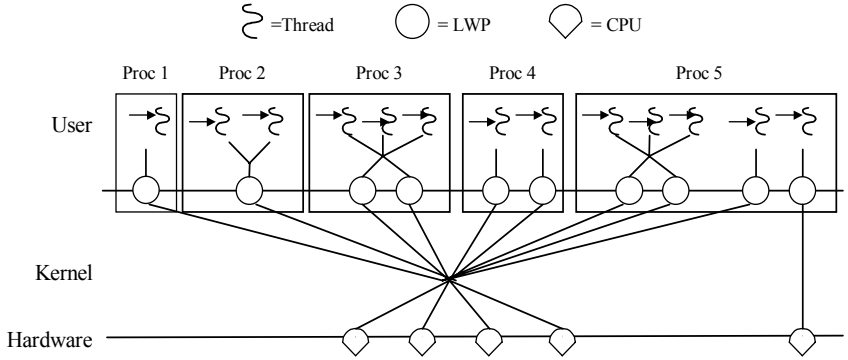
Supporto a thread a due livelli

Si comincia ad affermare l'idea di avere gestioni a due livelli dei thread
thread a **livello di applicazione**
thread a **livello di kernel**

L'utente deve specificare anche come le cose possono essere messe insieme

Solaris
Gli utenti definiscono i *processi applicativi*
Sono tenuti a definire per ciascun processo pesante
- i ***thread*** logici che li compongono
- gli ***lwp*** processori virtuali che richiedono

ed i legami con i processori disponibili

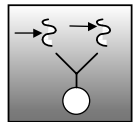


Lo schema consente di legare i thread applicativi in modi molto differenziati ai processori in esecuzione

Scheduling delle attività

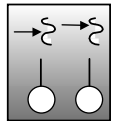
SunOS 5.x prevede molti possibili modelli di esecuzione

1) molti a uno



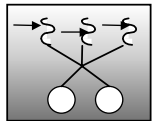
Più Thread utente schedulati su un unico LWP
primitive bloccanti -> blocco di tutto il processo
scheduling a livello utente
(HP-UX e DCE)

2) uno a uno



Un Thread utente per ogni LWP
primitive bloccanti -> blocco del thread
scheduling a livello kernel
IBM AIX, Microsoft Windows NT, IBM OS/2, LINUX

3) molti a molti

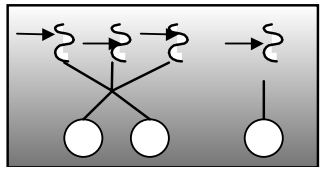


diversi Thread utente per ogni LWP
primitive bloccanti -> nessun blocco del thread
DEC Unix, Silicon Graphics IRIX

Scheduling con modello a due livelli

SUNOS 5.x

Solaris 2.x



scheduling a due livelli
con eventuale binding di un **thread** ad un **LWP**

Esigenze di **real-time**
in caso di **multiprocessore**

Particolarmente studiato per ottenere il numero di IWP necessari alla migliore esecuzione

In caso che non ce ne siano disponibili, possono essere creati automaticamente su necessità
se ci sono thread da eseguire
`thr_setconcurrency()`

Primitive di thread

API molto differenziate e poco portabili
non sono compatibili POSIX
che propone primitiva simili, ma diverse
(anche perdiverse versioni)

I thread hanno anche memoria comune tra loro o privata visibile a gruppi
oltre ai mutex, lock, condizioni, etc.

API thread

```
int thr_create (void *stack_base,
                size_t stack_size,
                void *(*start_routine) (void *),
                void *arg,long flags,
                thread_t *new_thread);
void thr_exit (void *status);
int thr_join (thread_t wait_for,
              thread_t *departed,
              void **status);
void thr_yield (void);
int thr_suspend (thread_t target_thread);
int thr_continue (thread_t target_thread);

int thr_keycreate (thread_key_t *keyp,
          void (*destructor) (void *value));
int thr_setspecific (thread_key_t key,
                     void *value);
int thr_getspecific (thread_key_t key,
                     void **valuep);

int thr_getconcurrency (void);
int thr_setconcurrency (int new_level);

int thr_getprio (thread_t target_th,int *pri);
int thr_setprio (thread_t target_th, int pri);
int mutex_init (mutex_t *mp, int type,
           void *arg);
int mutex_lock/mutex_unlock/mutex_trylock/
mutex_destroy (mutex_t *mp);

int cond_init, cond_wait, cond_signal,
    cond_broadcast , cond_timedwait
int sema_init, sema_wait, sema_trywait
etc.
```

Gruppi di Processi

Insieme di Processi che sono visti come una unica entità astratta per alcune azioni

Ad esempio: modello multi-server a multicast

Semantica di Gruppo

- un gruppo deve essere dinamico
- un processo può appartenere a più gruppi
- un gruppo deve essere riconosciuto in termini di comunicazione

Realizzazione attraverso il broadcast

Gruppi chiusi

solo i componenti possono inviare al gruppo

Gruppi aperti

la astrazione di gruppo è visibile ad ogni altra entità

Gruppi chiusi

per la realizzazione del parallelismo

Gruppi aperti

per la realizzazione di cliente/servitori multipli

Gruppi di Processi

Struttura ed organizzazione dei gruppi

Processi pari

tutti i processi sono pari

Processi gerarchici

presenza di un coordinatore e di coordinati

Appartenenza ad un gruppo

Un processo gestore

Esiste un processo che consente di entrare a fare parte di un gruppo e di lasciare il gruppo

Realizzazione Distribuita

tutti i processi mantengono la lista di appartenenza

Identificazione dei guasti distribuita

Scheduling dei Processi

Scheduler
allocazione dei processi al processore
Dispatcher
assegnamento del processore ad un processo

Lo scheduling è la parte che può essere distribuita
politica locale
politica globale

Scheduling Locale
Charlotte round-robin
V-kernel priorità
Accent 16 livelli di priorità variabile e time-slice

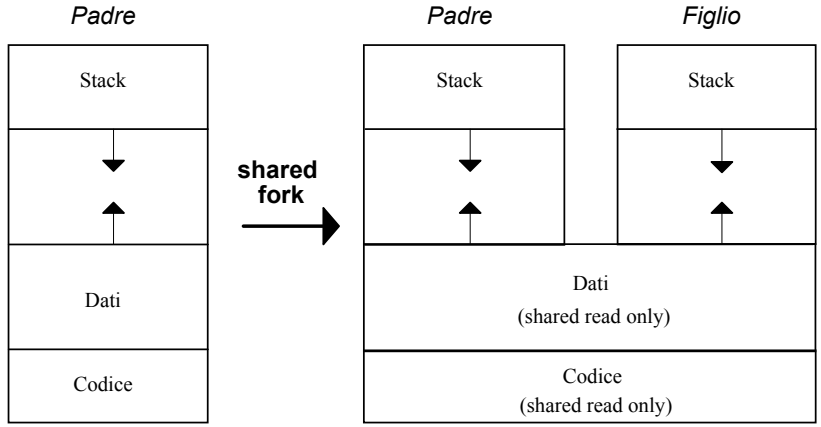
Necessità di *Scheduling Globale*
operazioni remote sui processi
meccanismi
gestione risorse remote
politiche

LOAD SHARING
utilizzo delle risorse in modo che nessun processore sia idle
LOAD BALANCING
bilanciare l'uso delle risorse per ottenere un carico equilibrato

Operazioni remote sui processi

creazione remota / terminazione remota
esecuzione remota

in alcuni sistemi è prevista una **fork remota**
e condivisione variabili
in SPRITE



Management delle **Risorse** del processo
file aperti
risorse di comunicazione

Gestione della creazione
le richieste (primitive) sono fatte a dei gestori che si coordinano tra di loro (e non al kernel locale)

Uso di gestori per la **trasparenza** del servizio
gestori *distinguono i servizi* ***locali*** *e servizi* ***remoti*** *fornendo una unica interfaccia all'utilizzatore*
I gestori costituiscono un livello di trasparenza

Esecuzione remota

possibilità di attivare un processo su un nodo diverso e di interagire
disponibile ai diversi livelli del sistema

In **V-kernel**: un comando usa altre workstation libere da carico locale

A livello utente, trasparenza meno
V-Kernel
<programma><args> @ <nomehost>
<programma><args> @ *
selezione esplicita od implicita dell'host

Problemi di Eterogeneità

- non esiste uno spazio globale dei nomi per le entità da riferire
- esistono diverse convenzioni per definire i servizi e gli attributi (sintassi dei comandi, etc.)
- è necessaria una trasformazione delle informazioni da uno spazio ad un altro

Requisiti

- necessità di propagare informazioni di stato dei processori
- non-interferenza con l'uso locale
- basso overhead della esecuzione remota

IMPLEMENTAZIONE

Distinguiamo **servitori** di calcolo e **clienti**
I GESTORI forniscono un servizio di **registrazione**

- i servitori si registrano come fornitori del servizio
- i clienti accedono alle informazioni

Due fasi:
Inizializzazione
La richiesta per la creazione di un nuovo processo segue i normali meccanismi
Si interessa un gestore remoto, anziché locale
Richiesta in **broadcast a più gestori**
Un gestore richiede al kernel la creazione e inizializzazione

Esecuzione
Non c'è differenza rispetto al caso locale:

- spazio inizializzato allo stesso modo
- ogni riferimento è indipendente dalla località
- kernel e gestori sono omogenei in locale o remoto

GESTIONE RISORSE

definizione di ***risorsa***

ogni componente riusabile o meno, sia hardware, sia software necessario alla applicazione o al sistema

Classificazione

- **risorse** basate sulla **astrazione**
  specifica (interfaccia visibile) e **implementazione**
- **risorse fisiche** **vs.** **risorse logiche**
  tutte distribuite
- **risorse di basso livello** **vs.** **risorse applicative**

Gestione organizzata in fasi (***statica e dinamica***)

- **pianificazione** della organizzazione e identificazione
  allocazione
  disponibilità
  costo
- **controllo** delle risorse (loro uso)
  controllo di accesso
  ottimizzazione
  autenticazione
  controllo di correttezza operazioni ed eccezioni

Realizzazioni

concentrate e **distribuite**

accuratezza delle informazioni

Criteri di performance

Delay ritardo nel produrre il servizio

service time
durata media di un servizio

waiting time
tempo di attesa per un servizio

response time
tempo di risposta per il completamento di un servizio

Tutti gli indicatori di tempo sono anche indicatori del carico del processore

throughput
servizi nell'unità di tempo (di osservazione) a diversi livelli

Si possono introdurre anche altri criteri legati alla gestione economica della risorsa

costo
tenere conto dei costi aggiuntivi di sistema conseguenti ad una azione di sistema

minima intrusione =>
limitare l'overhead del supporto

Descrizione delle risorse

attributi di una **risorsa**
- **classe** della risorsa
- **nome** della risorsa
- altri attributi correlati
  - **esecuzione** vincolata su una determinata macchina

con **fattori**
- qualità del servizio
- tempo di allocazione
- costo
- tempo di ritardo in comunicazione

gestione risorse
- *uso di servitore (complesso)* v*s. solo dati*
- *composte* v*s. multilivello*
- *mobili* v*s. immobili*

in caso di movimento e replicazione
- *con consistenza forte vs. con consistenza debole*

Controllo allocazione
- allocazione della risorsa e del nome

Controllo accesso
- locale vs. remoto

Condivisione delle risorse

Due modelli principali di condivisione delle risorse
- **service request**
- **file system distribuito**

Service Request
- **Servizio specifico** con **richiesta esplicita** dell'utente al servitore

modello Cliente/Servitore (C/S)

File System Distribuito (FSD)
- **Servizio unico** con **trasparenza** alla **allocazione delle risorse** usando **agenti**
- servizi completamente presenti, o solo in alcuni nodi (spesso detti ***file server***)

modello ad Agenti

Modelli di risoluzione del problema

Servizi (ed informazioni)

semplici	vs.	**multipli** (oggetti)
statici	vs.	**dinamici**
centralizzati	vs.	**distribuiti**

Modello a server

Un unico servitore (centralizzato o distribuito)

- per tutte le risorse
- che gestisce una sola classe di risorse

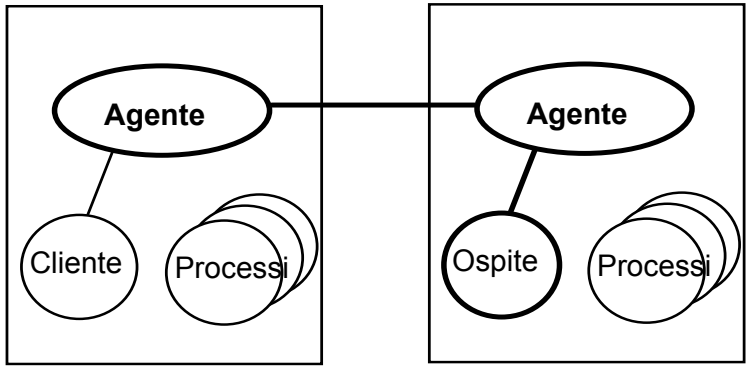
Uso di tabelle per le informazioni delle risorse

Affidabilità ==> Replicazione

Unico sistema di agenti per il servizio

agente per controllare e coordinare il servizio
in modo distribuito

Coordinamento tra agenti per l'uso delle risorse



Sono possibili fasi di **negoziazione** tra gli agenti prima dell'uso della risorsa
Con possibilità di **rifiuto**

LOAD SHARING

DETERMINARE dove e quando portare i processi da eseguire quando si presentano nel sistema

quali processi, quando, dove muovere

Implementazioni distribuite

alla ricerca di processori liberi in grado di essere utilizzati

Organizzazioni possibili ***STATICA vs. DINAMICA***

interconnessione

processori in ring logico	*statica*
processori in gerarchia logica MICROS	*dinamica*
processori liberi (worm)	*dinamica*

Ring logico

Ancora una struttura logica per la ricerca
con un token in un anello

Si usa un broadcast iniziale a tutti della richiesta di esecuzione
si distribuisce il carico secondo le risposte

struttura statica

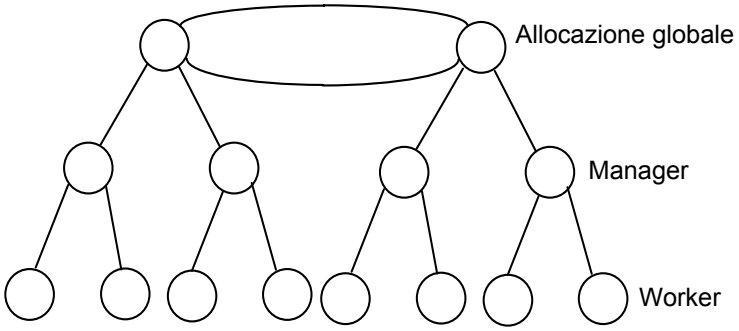
Gerarchia logica MICROS

MICROS sistema operativo distribuito

Obiettivi

- gestione di un numero di nodi molto elevato
- numero di utenti elevato e di applicazioni molto varie
- indipendenza dalla topologia
- gestione della replicazione

L'architettura è gerarchica
ma logica: non ci sono connessioni dirette



Worker ==> funzioni di calcolo ed I/O (**slave**)
Manager ==> funzioni di gestione

Numero di livelli dipende dal numero di worker

Guasti
master ==> più nodi master
slave ==> il master deve poter comandare i livelli sottostanti

Allocazione statica

Si allocano un numero di processori worker adatti ed i relativi manager

Allocazione dinamica

Si possono richiedere altri nodi di elaborazione nella gerarchia
al limite si possono chiedere nuove risorse al livello sovrastante

Worm

Approccio nuovo

- parallelo
- tollerante ai guasti
- adattativo al rete ed alla topologia
- per un bilanciamento nell'uso delle risorse

Worm/verme fatto di uno o più segmenti, cioè processi che possono anche comunicare tra loro

Un verme si incarica di cercare i nodi liberi attraverso **clonazione** su nodi liberi
uso di **probe** mandati dai segmenti che vogliono espandersi
Metodologia decentralizzata
NON si conosce lo stato del verme globalmente

Una copia del verme per nodo

Applicazione sta al disopra

LOAD BALANCING

Fatto durante l'esecuzione

migrando elementi anche dopo che hanno eseguito

RISORSE e CARICO - Modelli **statici**

risorse predeterminate

RISORSE e CARICO - Modelli **dinamici**

risorse create dinamicamente e non prevedibili

Allocazione iniziale di **processi**

risorse logiche statiche

struttura statica

Creazione successiva di **processi**

risorse logiche dinamiche

struttura dinamica

Approccio

statico	compilazione e ottimizzazione
dinamico	run-time
misto	entrambi gli approcci

LOAD BALANCING

MIGRAZIONE => OBIETTIVI

- uso delle risorse **più CORRETTO ed EFFICIENTE**
  disomogeneità delle risorse fisiche
  file partizionati e presenti su nodi specifici
- **BILANCIAMENTO del carico computazionale**
  divisione del carico tra i nodi secondo necessità
- **DINAMICITÀ e MOBILITÀ**
  possibilità di fare fronte a una allocazione anche non ottimale, o non più ottimale
- **TOLLERANZA ai guasti**
  una risorsa può essere trasferita in altro nodo prima del crash totale

Requisiti

Performance	buon uso delle risorse
Efficienza del sistema	overhead limitato
Trasparenza o meno al livello applicativo	gestione automatica

Eterogeneità

diverse architetture e modelli computazionali

caratteristiche dell'ambiente

- capacità di osservazione (monitor)
- affidabilità
- dinamicità

MONITORING

Identificazione del carico del sistema usando osservazioni sul carico corrente

assumendo *continuità della applicazione* e *gradienti limitati*

Raccolta di informazioni di carico su processori, risorse e comunicazione

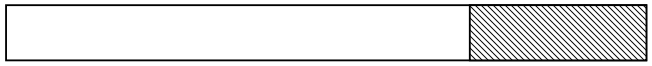
- ad eventi
- osservazioni su intervallo limitato
- dati statistici

*Le informazioni monitorate sono usate per la **previsione** delle variazioni del carico nel futuro*

Necessità di **limitare le informazioni** da osservare e mantenere

per limitare intrusione

Percentuale di occupazione delle risorse



Lavoro applicativo

Supporto per sistema

PRINCIPIO di NON INTRUSIONE

i livelli di supporto di sistema non devono apportare una variazione sostanziale nell'applicazione
Si deve quindi assolutamente limitare il loro overhead

Altri requisiti per la migrazione

- **Pre-emption**
  priorità nell'uso locale
- **Evitare dipendenza residue**
  lasciare traccia nel sistema delle migrazioni può diventare costoso
- **Evitare thrashing**
  movimento di un solo processo che non riesce a proseguire
- **Migrazioni multiple**
  consentire concorrenza nella migrazione per parallelizzare: gli strumenti più complessi

Indipendenza dei meccanismi dalle politiche
  livelli di kernel che garantiscono scambio dei messaggi devono avere poca interferenza

Affidabilità
  per i protocolli di migrazione

Trasparenza (?)
  senza alcuna modifica del codice applicativo

MIGRAZIONE DI DATI

alcuni sistemi considerano altre **entità**
candidate alla migrazione

FILE spostati da un nodo ad un altro

LOCUS (1981)
estensione a UNIX

OBIETTIVI
mobilità di processi e di file
replicazione dei file e
consistenza dati a fronte di guasti

ACCESSO **TRASPARENTE** ai FILE
TRASPARENZA della REPLICAZIONE

In sistemi con modelli computazionali diversi
si possono considerare altri candidati

MIGRAZIONE DI **OGGETTI**
MIGRAZIONE DI **AGENTI**

In caso di migrazione
e durante la migrazione

PROBLEMA dei MESSAGGI in fase di consegna
messaggi da spedire a chi sta migrando

Cambiamento di nome della risorsa mobile

strategie pessimiste/proattive

- **Ridirezione dei messaggi**
  bufferizzazione dei messaggi arrivati per il processo da parte del kernel, che riceve poi i messaggi
  Il nodo di partenza tiene traccia della allocazione del processo; i clienti mandano al vecchio nodo

- **Riqualificazione dell'allocazione**
  i messaggi arrivati per il processo sono mantenuti ma i clienti sono informati della nuova allocazione
  Si mandano messaggi a tutti i potenziali clienti
  Il nodo di partenza tiene traccia della allocazione del processo solo fino al completamento del trasferimento

strategie ottimiste/reattive

- **Recovery da parte dei clienti**
  Non si mantiene la nuova allocazione del processo e non si informano i clienti
  Il messaggio può fallire: è compito del cliente di ritrovare la nuova allocazione

MIGRAZIONE DI AGENTI

nei modelli di **movimento con agenti**
definiamo **attività in cui il movimento** è un requisito
l'obiettivo non è legato al bilanciamento del carico o a considerazioni di uso di risorse

derivato da specifiche precise di applicazione
sistemi mobili e geografici con coordinamento
sistemi globali (basati su **Web** ed **Internet**)

criteri

- gli agenti devono muoversi anche ritornando su *nodi già visitati* (per riportare informazioni trovate)
- non ci sono *vincoli di costo* nella decisione di migrazione, fatta in ogni caso
- necessità di realizzare *meccanismi efficienti*

Esempi di applicazioni

network management non cliente servitore (vedi snmp o CMIP OSI): uso di agenti che propagano ai diversi nodi le esigenze e raccolgono informazioni

supporto di ricerche per **informazioni su web**: sistemi di caching di dati, ricerca più intelligente di informazioni, gestori per verificare esistenza di siti prima che un utente vi acceda (commercio elettronico)

sistemi di lavoro **cooperativo** (**C**omputer **S**upported **C**ooperative **W**ork): uso di agenti per il coordinamento necessario per ottenere viste ed eventi comuni