Set up di carichi commerciali per la valutazione di architetture multiprocessore non convenzionali

Michele Campagni

Indice

CAPITOLO 1

Introduzione

Per le cache integrate nei moderni microprocessori si ipotizza attualmente un tempo di accesso uniforme. In realtà, se in futuro i produttori di microprocessori seguiranno la tendenza attuale di aumentare la frequenza di lavoro, questo modello si scontrerà con il limite fisico del *wire-delay*. Il wire-delay è il ritardo alla propagazione del segnale introdotto dai canali di collegamento e fissa dei limiti precisi oltre i quali un modello che consideri uniformi i tempi di accesso alle cache non può funzionare. Considerando la velocità della luce nel silicio[1], il tempo di attraversamento di un attuale Pentium 4, il cui *core* misura 13.97 mm, è 0.15875 ns. Il reciproco di questo tempo di attraversamento, circa 6.3 GHz, fissa la massima frequenza di lavoro al di sopra della quale un segnale impiega più di un ciclo di clock per attraversare l'intero processore. La frequenza che abbiamo calcolato ora è confrontabile con quelle degli attuali microprocessori per cui il problema si presenterà in tempi brevi. In figura 1.1 vediamo un moderno processore; la cache, rappresentata in rosso a destra della figura, occupa una vasta area e le zone più vicine ai bordi del chip richiedono tempi di accesso maggiori rispetto a quelle vicine al controller della memoria.

I modelli di cache che considerano tempi di accesso non uniformi prendono il nome di NUCA[2].

Un'altra strada per incrementare le prestazioni è l'introduzione di processori *multi-core*. Alcune implementazioni di questi chip condividono la cache che si trova normalmente disposta al centro come in figura 1.2. I dati contenuti nella cache devono avvicinarsi il più possibile al processore che li riferisce più di frequente, considerando però che potrebbe non essere l'unico. Il problema si estende

[1] circa 88 milioni di metri al secondo

[2] Non-Uniform Cache Architecture

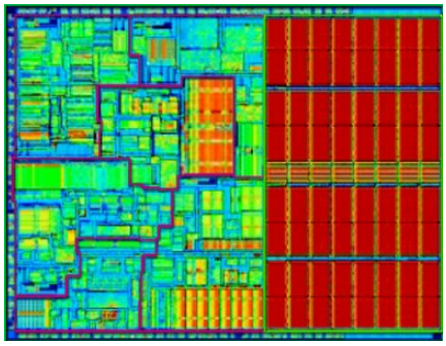

Figura 1.1: Il core di un moderno processore

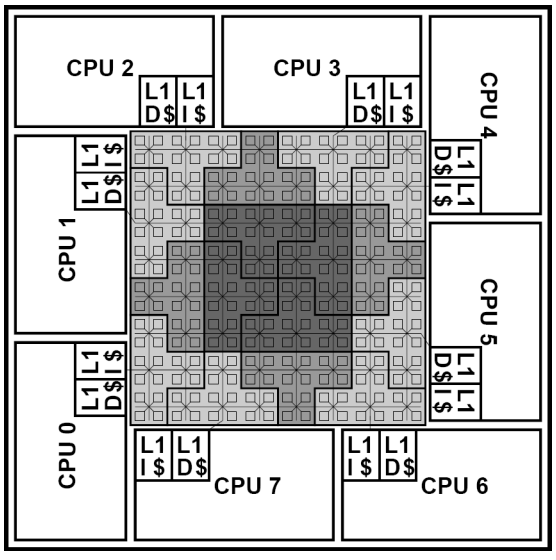

Figura 1.2: Condivisione della cache in sistemi multi-core[2]

anche allo schedulatore del sistema operativo che deve essere in grado di valutare se convenga eseguire la migrazione dei processi sui vari processori piuttosto che aspettare un avvicendamento dei dati della cache.

Se le cache di tipo NUCA saranno la prossima, temporanea, soluzione allo scontro tra la fisica e l'incremento delle prestazioni dei calcolatori, è importante conoscere quale sarà l'impatto dell'introduzione di queste cache negli attuali sistemi di elaborazione. Le strade per valutare questo impatto sono, come si può immaginare, due: costruire l'hardware necessario oppure simularlo.

Ammesso che la riprogettazione di un sistema dotato di cache NUCA si limiti al solo processore, la produzione dei prototipi hardware ha un costo molto elevato. Per questo motivo, come per molti campi dell'ingegneria, esistono dei simulatori che permettono di valutare preliminarmente le prestazioni di questi nuovi sistemi.

Una volta risolto il problema della disponibilità di un sistema da testare, simulato o reale, resta da chiedersi quale sia un benchmark valido per verificare i reali guadagni di prestazioni.

CAPITOLO 2

Obiettivi

In questo lavoro ci proponiamo di configurare un ambiente di simulazione per la valutazione delle prestazioni delle architetture multiprocessore con carichi commerciali. L'ambiente realizzato si presenta sotto forma di dump di dischi che possono essere copiati su hard disk fisici o logici ed eseguiti su macchine reali con processori Sparc oppure su piattaforme di simulazione come Simics. Le immagini dei dischi permettono il boot di un sistema Solaris con Oracle configurato e pronto ad eseguire le query del benchmark TPC-H. I vari sistemi sono stati configurati con lo scopo di concentrare il carico di esecuzione sulle cache piuttosto che sui sistemi disco in quanto l'ambiente deve permettere di valutare le prestazioni del sottosistema di memoria.

CAPITOLO 3

Simics

3.1 Presentazione

Simics[14] è un simulatore di architetture capace di emulare sistemi con fedeltà sufficiente da permettere il boot di sistemi operativi e l'esecuzione dei relativi programmi. Simics permette di definire configurazioni di sistemi quali Sun, Alpha, Mips, e X86. Tramite un linguaggio di tipo script è possibile configurare backplanes e schede, memoria, dischi e cpu e avviare il boot di un sistema operativo quale Unix o Windows. In tabella 3.1 sono riportate le architetture simulate da Simics.

Simics non è un simulatore microarchitetturale ma l'università del Wisconsin ha prodotto GEMS[1][13], un modulo che rende possibile la definizione microarchitetturale di memorie, memorie cache e processori out-of-order. GEMS è diviso in Opal che è un modulo che permette di dettagliare modelli di processori out-of-order a livello di branch predictor, numero di pipeline, ecc. e Ruby che permette di definire gerarchie di memorie. Gems permette, tramite il linguaggio SLICC[2], di specificare i protocolli di coerenza delle cache. I modelli di cache che Gems supporta sono i seguenti[3][8]:

- **MOSI_SMP_bcast**: a snooping, broadcast-based SMP cache coherence protocol and system. Each processor has a L1I, L1D, and unified L2 cache. The same SLICC controller is used for all caches (MOSI_SMP_bcast-cache.sm).

[1] General Execution-driven Multiprocessor Simulator
[2] Specification Language including Cache Coherence
[3] non tradotte per non perdere leggibilità

Architettura	**Sistemi Simulati**	**CPU Simulate**
Alpha	AlphaPC 164 (EV5) AlphaPC 264DP (EV6)	Alpha 21164 Alpha 21264
ARM	Assabet	Intel StrongARM ARM966E-S
IA-64	Itanium system based on the 460GX chipset	Itanium Itanium 2
MIPS	MIPS Malta	MIPS-4Kc MIPS-5Kc PMC E9000 QED RM7000
PowerPC	Artesyn PM/PPC IBM Walnut (PPC 405 GP) IBM Ebony (PPC 440 GP)	IBM PowerPC 403GCX IBM PowerPC 405GP IBM PowerPC 440 GP/GX IBM PowerPC 750 (FX/GX) IBM PowerPC 755 IBM PowerPC 970FX Freescale PowerPC 7400 Freescale PowerPC 7447 Freescale PowerPC 7450 Freescale PowerPC 7457 Freescale PowerPC 603e Freescale PowerPC 8260 Freescale PowerPC 8270 Freescale PowerPC 8280 Freescale PowerPC 8540 Freescale PowerPC 8641D
SPARC	SunFire (Sun Enterprise 3500/4500/5500/6500 server) Serengeti (Sun Fire 3800/4800/6800 server) Fiesta (Sun Blade 1500)	UltraSPARC II UltraSPARC III UltraSPARC III Cu UltraSPARC IV (+)
TI DSP		TMS320C64
x86	Standard PC	486 SX Pentium Pentium/MMX Pentium Pro Pentium II Pentium III Pentium 4
AMD64/EM64T	Standard PC Tyan Tomcat K8S (AMD-8111-based system)	AMD Opteron AMD Athlon 64

Tabella 3.1: Architetture simulate da Simics

- **MOESI_SMP_directory**: a directory-based SMP cache coherence protocol and system. Each processor has a L1I, L1D, and unified L2 cache. The same SLICC controller is used for all caches.
- **MOESI_SMP_hammer**: a SMP system modeled after the AMD Opteron. The network can be unordered.
- **MSI_MOSI_CMP_directory**: a two-level directory protocol for Chip-Multiprocessors. The L1 and L2 controllers are split, and the L2 cache is shared by all processors on the same chip. Inclusion is maintained between L2s and the L1s, and a sharers list is kept in each L2 cache line.
- **MOESI_CMP_directory**: a two-level directory protocol for Chip-Multiprocessors. Non-inclusive L1/L2 caching with blocking caches. Typically exhibits higher performance than MSI_MOSI_CMP_directory however is more complex and may still have rare protocol bugs
- **MOESI_CMP_token**: a CMP system based on Token Coherence (TokenCMP).
- **MOSI_SMP_bcast_1level**: similar to MOSI_SMP_bcast except that each processor has a single unified cache
- **MOSI_SMP_directory_1level**: a directory protocol with only a single unified cache per processor
- **SNUCA**: static placement of cache blocks
- **DNUCA**: dynamic placement of cache blocks allowing blocks to migrate towards requesting processors
- **RNUCA**: dynamic placement of cache blocks allowing blocks to migrate and replicate towards requesting processors

Simics è scaricabile per la valutazione[4] previa registrazione. La licenza d'uso accademica permette a ricercatori e membri delle università di scaricare ed utilizzare tutte le versioni disponibili. Sono disponibili anche interi dump dei dischi di sistemi con GNU/ già installato. Per motivi di licenza non sono disponibili dump con Solaris installato, pertanto, per questa installazione, è necessario procurarsi i dischi di installazione di Solaris nella versione scelta.

[4] `https://www.simics.net/evaluation/form_dl.php`

3.2 Velocità di emulazione

Allo stato attuale Simics non è multiprogrammato il che vuol dire che anche se eseguito in un sistema multiproceossore, Simics gira su un solo processore della macchina host. Supponendo di voler emulare un sistema con 8 processori, un solo processore della macchina host dovrà farsi carico di emulare non solo gli 8 processori della macchina target ma anche l'overhead introdotto dalle comunicazioni fra i processori emulati.

Inoltre va considerato che Simics implementa il *full system simulation*, che implica la creazione di un modello software per ogni modulo hardware emulato. L'hardware viene emulato a livello di registri ed interrupt; a questo livello il software che viene eseguito sul sistema emulato non è in grado di riconoscere differenze con l'hardware reale. Di questa ulteriore emulazione, molto importante perché permette di eseguire il software sulle macchine simulate senza alcuna modifica, se ne fa carico il solito processore della macchina host.

L'Università del Wisconsin ha effettuato uno studio per la valutazione delle prestazioni di Simics + Gems, emulando con un AMD Opteron da 2.2 GHz una macchina UltraSPARC con 8 processori. Come consigliato da Virtutech, il sistema host e quello target avevano la stessa quantità di memoria ram, questo per evitare rallentamenti per accessi allo swap file. I risultati sono riportati in tabella 3.2.

	Time Target	**Slowdown**	**Slowdown/CPU**
Target	20 ms	1 x	1 x
Simics	1 minuto	3000 x	380 x
Simics + Ruby	15 minuti	45000 x	5600 x
Simics + Ruby + Opal	45 minuti	140000 x	17000 x

Tabella 3.2: Slowdown di Simics + Gems rispetto ad un target reale

Confrontato con una Workstation Sun Ultra 10 reale[5], Simics impiega in simulazione funzionale e a parità di numero di processori e ram circa 15 minuti per eseguire un processo che sulla macchina reale gira in 30 secondi. Usando la simulazione microarchitetturale (Simics + Ruby + Opal) questo divario aumenta enormemente passando da 15 minuti a 20 ore, un rapporto fra la macchina reale e quella simulata di 1/2400. Raddoppiare il numero di processori simulati vuol dire dimezzare questo rapporto che diventa, nel caso di 8 processori, 1/19200. Un secondo richiede una simulazione di 5 ore e 20 minuti. In verità questi confronti non hanno molto significato in quanto nel passaggio dal modello funzionale a quello microarchitetturale si introducono profondi cambiamenti nei rapporti fra i moduli del sistema. Uno di questi cambiamenti si traduce in un maggior numero

[5]sistema con un processore Sparc da 333 MHz con 1 GB di ram

di istruzioni generate per l'accesso al bus della memoria che porta a sperimentare esplosioni del numero di istruzioni a parità di compiti assegnati al processore.

Per avere un'idea dello slowdown che si produce introducendo Ruby + Opal basti pensare che l'esecuzione del comando unix 'ls' in una macchina simulata con 8 processori impiega circa 3.5 ore solo per cominciare a presentare un risultato (4 ore per stampare una directory di circa 50 file).

3.3 Configurazione

Fra le architetture simulate, Simics mette a disposizione una macchina Sun, la cui configurazione è chiamata Peanuts[15]. Peanuts è basata sul modello reale di una Sun Enterprise Server 6500 le cui caratteristiche sono illustrate in tabella B.2.

Peanuts dispone di script per l'installazione di Solaris™ 10 e quest'ultimo viene distribuito da Sun sotto forma di immagini ISO[6] 9660. La macchina simulata si chiama convenzionalmente *target* mentre la macchina su cui viene lanciata la simulazione, *host*. Peanuts è configurata con una scheda grafica pgx64 necessaria per l'avvio del programma di installazione di Oracle. Sono necessarie le modifiche che vedremo per assicurare ad Oracle uno spazio sufficiente per l'installazione del DBMS[7] e del database. All'avvio Simics presenta la seguente schermata:

```
Checking out a license... done: academic license.
Looking for additional Simics modules in ./modules

  +----------------+    Copyright 1998-2005 by Virtutech, All Rights Reserved
  |   Virtutech    |    Version: simics-2.2.19
  |     Simics     |    Compiled: Tue Aug 16 20:22:30 CEST 2005
  +----------------+
    www.simics.com      "Virtutech" and "Simics" are trademarks of Virtutech AB

Type 'copyright' for details on copyright.
Type 'license' for details on warranty, copying, etc.
Type 'readme' for further information about this version.
Type 'help help' for info on the on-line documentation.

simics>
```

3.3.1 OpenBoot

OpenBoot è un *firmware*[8] hardware-indipendent sviluppato da Sun Microsystems, in altre parole l'equivalente del BIOS[9] dei sistemi x86. OpenBoot è più

[6] International Organization for Standardization

[7] Database Management System

[8] programma che carica il sistema operativo, generalmente contenuto in una memoria di sola lettura

[9] Basic Input/Output System - Basic Integrated Operating System

di un semplice programma di configurazione dell'hardware e mette a disposizione una shell basata sul linguaggio di programmazione Forth. Con OpenBoot si può configurare l'hardware, specificare il disco da usare per l'avvio del sistema operativo e passare a Solaris dei parametri di avvio (vedi paragrafo A.2.2). Fra questi parametri ci sono quelli legati alla redirezione input/output; è possibile specificare di usare una delle porte seriali come *console* per Solaris.

Sulle macchine Sparc si accede ad OpenBoot in qualunque momento mediante la combinazione di tasti `Stop-A`[10], la tastiera Sun differisce da quella dei sistemi x86 per la presenza di una ulteriore fila di tasti a sinistra, come si può vedere nella figura B.3.

```
ok help
go                      - Resume execution of OS
boot                    - Boot the default OS
boot <device-name>      - Boot from the specified disk
printenv                - Display all configuration variables
setenv <name> <value>   - Set a configuration variable
devalias                - Display all device aliases
devalias <name> <value> - Create or change a device alias
show-devs               - Display the names of all devices

Many other commands are available.
ok
```

Figura 3.1: Esempio di schermata di OpenBoot

Simics fornisce un'interfaccia OpenBoot che permette di accedere ai più importanti parametri, di seguito riportati:

```
1  simics> system.get-prom-env
2  diag-file            :
3  ttya-rts-dtr-off     : false
4  diag-switch?         : true
5  scsi-initiator-id    : 0x7
6  screen-#rows         : 0x18
7  memory-interleave    : max
8  input-device         : ttya
9  silent-mode?         : false
10 auto-boot?           : true
11 ttya-mode            : 9600,8,n,1,-
12 boot-device          : disk diskbrd diskisp disksoc net
13 hardware-revision    :
14 mfg-mode             : off
15 fcode-debug?         : false
16 nvramrc              :
17 ttyb-rts-dtr-off     : false
18 watchdog-reboot?     : false
19 local-mac-address?   : false
20 powerfail-time       : 0x0
21 diag-device          : disk diskbrd diskisp disksoc net
```

[10]notare che il tasto `Stop` non esiste nelle tastiere convenzionali

```
22  security-#badlogins  : 0x0
23  #power-cycles        : 0x0
24  output-device        : ttya
25  oem-banner           :
26  keyboard-click?      : false
27  sbus-specific-probe  :
28  oem-banner?          : false
29  boot-file            :
30  load-base            : 0x4000
31  security-mode        : none
32  sbus-probe-default   : 3120
33  last-hardware-update :
34  ttyb-ignore-cd       : true
35  disabled-board-list  :
36  use-nvramrc?         : false
37  oem-logo?            : false
38  security-password    :
39  ansi-terminal?       : true
40  configuration-policy : component
41  oem-logo             :
42  screen-#cols         : 0x50
43  ttyb-mode            : 9600,8,n,1,-
44  keymap               :
45  ttya-ignore-cd       : true
46  disabled-memory-list :
47  boot-command         : boot disk1 -v
48  diag-level           : max
```

Input-device (riga 8) ed output-device (riga 24) specificano quale console deve usare Solaris. Con `ttya` si specifica di usare la prima porta seriale; per tornare alla visualizzazione su console grafica occorre impostare:

```
input-device          : keyboard
output-device         : screen
```

mediante i comandi:

```
system_cmp0.set-prom-env input-device "keyboard"
system_cmp0.set-prom-env output-device "screen"
system_cmp0.set-prom-env boot-command "boot disk1 -rv"
```

L'ultimo comando serve per obbligare Solaris a riconfigurare le impostazioni della console. Gli oggetti istanziati (porta seriale o console grafica) devono essere stati prima creati come oggetti di Simics. Per una lista dei componenti oggetto di Simics si rimanda all'appendice B.

3.3.2 Configurare il numero di processori e la ram

Simics può essere richiamato da shell, specificando un file di configurazione che contiene la descrizione della macchina, la configurazione del backplane, delle interfacce e la dimensione della memoria:

```
1   @if not "boards" in dir():
2           boards = [[4, 2, 1024],[5,2,1024],[8,2,1024],[9,2,1024]]
3
4   # do auto-login per default
5   @if not "do_login" in dir():
6           do_login = 1
7
8   @donut_disk_size = 8513945600L
9   @donut_files = [["disco1.craff", "ro", 0, 8513945600L]]
10
11  # the rest is common for all donut machines
12  run-command-file "donut-common.simics"
```

La riga 2 specifica una configurazione con 8 processori; la matrice specificata nel file di configurazione:

$$\begin{pmatrix} 4 & 2 & 1024 \\ 5 & 2 & 1024 \\ 8 & 2 & 1024 \\ 9 & 2 & 1024 \end{pmatrix}$$

deve essere letta per righe, ognuna delle quali identifica una scheda CPU (figura B.2). La prima colonna identifica il numero d'ordine dello slot di espansione nel quale è installata la scheda. La seconda colonna e la terza colonna indicano rispettivamente il numero di processori installati e la quantità di memoria installata nella scheda CPU. Sommando le CPU e la ram otteniamo una macchina con 8 processori e 4 GB di ram.

3.3.3 Configurare la velocità dei processori

La frequenza dei processori simulati può essere settata arbitrariamente, tuttavia questo parametro non influisce sulla velocità di simulazione bensì il numero di istruzioni che vengono eseguite prima che si consideri passata un'unità di tempo simulato.

Per questo, quando vengono eseguiti programmi che fanno uso di interruzioni periodiche, come nel caso delle interfacce grafiche, tenere alto questo valore può aumentare i tempi di risposta dei commandi immessi da tastiera e da mouse, pur non comportando alcun aumento di prestazioni della macchina target.

Simics assume che per ogni ciclo di clock venga eseguita un'istruzione per ogni processore per cui, supponendo che il driver del mouse controlli lo stato della periferica ogni 10 millisecondi, avere settato la velocità del processore a 200 MHz piuttosto che a 100 MHz significa dover aspettare che vengano simulate il doppio delle istruzioni prima che l'interrupt passi il controllo al driver del mouse.

3.3.4 Aggiungere un disco SCSI

Abbiamo scelto di aggiungere un disco SCSI[11] alla configurazione della macchina target per installare Oracle su un disco indipendente.

Antepongo un minimo di terminologia per introdurre i parametri che useremo per la configurazione del nuovo disco. La struttura logica di un hard disk è quella riportata in figura 3.2. Un hard disk è composto da uno o più dischi ed una o più testine (o *head*, in genere le testine sono in numero doppio rispetto ai dischi) ed è suddiviso in cilindri, tracce e settori. I produttori di hard disk riportano sull'etichetta il numero di cilindri, settori e testine[12].

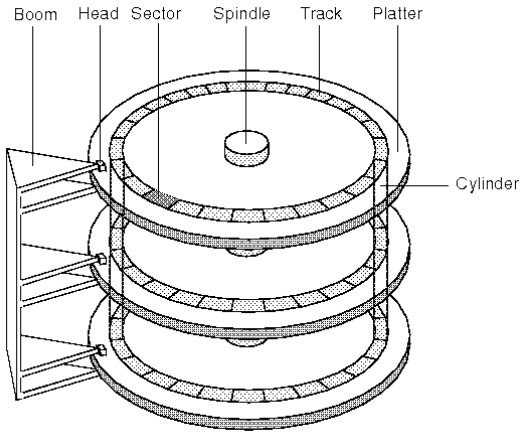

Figura 3.2: Geometria di un hard disk

Per far riconoscere a Simics/Solaris un nuovo disco scsi è necessario installarlo virtualmente configurando la macchina target:

```
1  script-branch {
2      wait-for-variable machine_defined
3      local \$disk = {create-std-scsi-disk size = 19998720000L)
4      connect-components $scsi_bus scsi-slot-2 $disk scsi-bus
5  }
6  run-command-file peanut-common.simics
```

Questo script di configurazione antepone al caricamento della configurazione di default di Simics (linea 6) la creazione di un disco di dimensioni di circa 20 GB collegato al bus scsi. *Scsi-slot-2* si riferisce all'id del disco nella catena scsi. Il comando connect-components configura il collegamento del disco virtuale al backplane della macchina simulata.

[11] Small Computer System Interface

[12] per ragioni di compatibilità questa geometria può subire una rimappatura per limitare la grandezza di uno o più determinati parametri al limite massimo supportato dai vecchi BIOS

Questo collegamento non è sufficiente a far vedere il nuovo disco a Solaris; occorre creare prima una partizione vuota e la geometria della *partition table*[13] deve corrispondere a quella fisica che abbiamo imposto:

```
simics> sd1.create-sun-vtoc-header 38752 16 63
simics> sd1.create-sun-vtoc-partition number = 0 start-block = 0
  num-blocks = 39060000 flag = RW tag = root
```

Il *num-blocks* è stato ricavato con la formula $((38752-2)*16)*63)$. La dimensione totale del disco è $39060000 * 512 = 19998720000$. A questo punto occorre dire a Openboot di riconfigurare Solaris per riconoscere il nuovo disco installato:

```
simics> system_cmp0.set-prom-env boot-command "boot disk1 -rv"
```

Al boot di Solaris il disco viene riconosciuto ma ancora non ha un filesystem che va creato con il comando:

```
# newfs /dev/dsk/c0t2d0s0
```

a questo punto il disco è pronto per essere collegato al mountpoint:

```
# mkdir /disk
# cat >> /etc/vfstab
/dev/dsk/c0t2d0s0  /dev/rdsk/c0t2d0s0  /disk ufs  2  yes    -
# mount /disk
```

Non resta che riavviare Solaris con:

```
# init 0
```

e salvare la nuova configurazione di Simics:

```
simics> system_cmp0.set-prom-env boot-command "boot disk1 -rv"
simics> save-persistent-state new-disk1.state
```

Per poter usare la nuova configurazione basta caricare Simics e dare il comando:

```
simics> load-persistent-state new-disk1.state
```

3.4 Gems

Gems[11] è composto da Ruby e Opal che sono distribuiti come due moduli di Simics. Ruby modella i sistemi di memoria adatti ad architetture multiprocessore; Opal implementa in dettaglio i processori *out of order*.

Alla data della pubblicazione di questa tesi, GEMS è in grado di integrarsi solo con Simics 2.2.19 mentre Simics è giunto alla versione 3. Per installare GEMS occorre scomprimerlo in una directory che chiameremo '$GEMS' e compilarlo.

Gems è disponibile presso *http://www.cs.wisc.edu/gems*, una volta scaricato va scompattato nella directory che abbiamo scelto e vanno creati i link appositi:

[13] la tabella di allocazione delle partizioni di un disco fisso

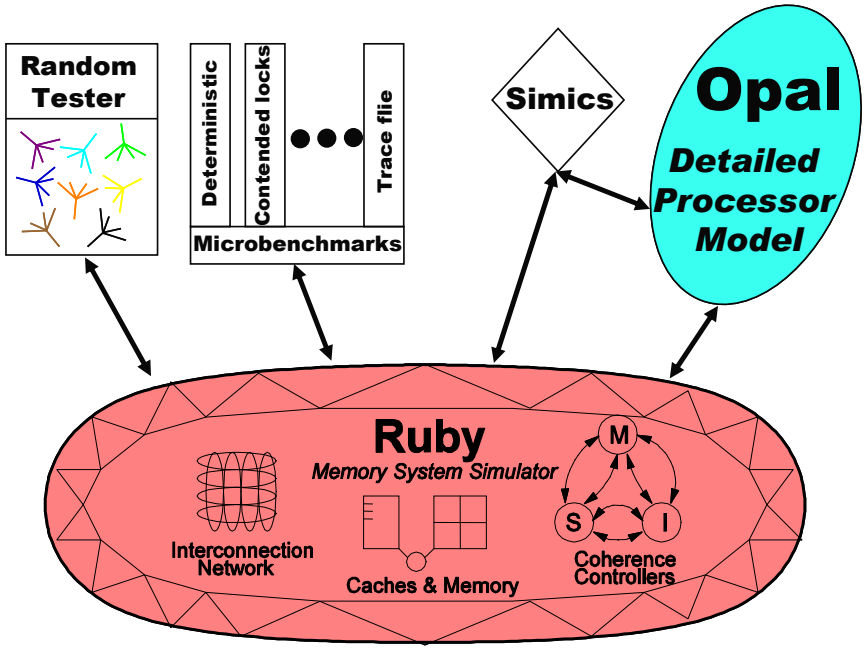

Figura 3.3: I moduli Ruby e Opal

```
1   cd $GEMS
2   tar xvzf gems-release1.0.tar.gz
3   tar xvzf simics-2.0.25-x86-linux.tar.gz $GEMS
4   ln -s $GEMS/simics-2.0.25 $GEMS/simics
```

A questo punto si configura Simics e si predispongono i link per il corretto funzionamento dei moduli Ruby e Opal:

```
5   cd $GEMS/simics/x86-linux
6   ../configure
7   cd $GEMS/simics/src/extensions/
8   mkdir ruby
9   cd ruby
10  ln -s ../../../../ruby/module/ruby.c
11  ln -s ../../../../ruby/interfaces/mf_api.h
12  ln -s ../../../../ruby/module/Makefile
13  ln -s ../../../../ruby/module/commands.py
14  ln -s ../../../../ruby/simics/commands.h
15  cd ..
16  mkdir opal
17  cd opal
18  ln -s ../../../../opal/module/opal.h
19  ln -s ../../../../opal/module/opal.c
```

```
20  ln -s ../../../../opal/module/Makefile
21  ln -s ../../../../opal/system/hfa_init.h
22  ln -s ../../../../opal/module/commands.py
```

È necessario editare il file *$GEMS/simics/config/modules.list-local* ed aggiungere le seguenti linee:

```
ruby | API_2.0 | v9
opal | API_2.0 | v9
```

Le libstdc++ e le libgcc_s vanno copiate in *$GEMS/simics/x86-linux/sys/lib*:

```
23  cp /usr/lib/libstdc++.so.5 $GEMS/simics/x86-linux/sys/lib
24  cp /lib/libgcc_s.so.1 $GEMS/simics/x86-linux/sys/lib/
```

Vanno editati i files *$GEMS/common/Makefile.common* per settare la giusta versione di Simics ed il path del compilatore C che si intende usare e *$GEMS/ruby/-module/Makefile* per settare il path delle librerie al runtime nel caso in cui non si usi il compilatore nativo.

Possiamo quindi lanciare la compilazione di Ruby ed Opal:

```
25  cd $GEMS/ruby
26  make PROTOCOL=MSI_MOSI_CMP_directory
27       DESTINATION=MSI_MOSI_CMP_directory
28  cd $GEMS/opal
29  make module DESTINATION=MSI_MOSI_CMP_directory
30  export SIMICS_EXTRA_LIB="./modules"
```

Questa procedura si riferisce all'installazione di GEMS su macchine GNU/Linux x86.

3.4.1 Istanziare Gems

Dopo aver avviato Simics ed aver caricato un checkpoint o una configurazione occorre usare questi comandi:

```
1  istc-disable
2  dstc-disable
3  instruction-fetch-mode instruction-fetch-trace
4  load-module ruby
5  load-module opal
6  ruby0.setparam g_NUM_PROCESSORS 4
7  ruby0.setparam g_PROCS_PER_CHIP 4
8  ruby0.setparam g_NUM_MEMORIES 8
9  ruby0.setparam_str g_NETWORK_TOPOLOGY FILE_SPECIFIED
10 ruby0.setparam_str g_DYNAMIC_TIMEOUT_ENABLED false
```

```
11  ruby0.setparam_str REMOVE_SINGLE_CYCLE_DCACHE_FAST_PATH true
12  ruby0.setparam_str g_CACHE_DESIGN NUCACOL
13  ruby0.setparam_str g_adaptive_routing false
14  ruby0.setparam NUMBER_OF_VIRTUAL_NETWORKS 7
15  ruby0.setparam g_endpoint_bandwidth 1000
16  ruby0.setparam_str g_NUCA_PREDICTOR_CONFIG SNUCA
17  ruby0.setparam g_NUM_L2_BANKS 32
18  ruby0.init
19  opal0.init
20  opal0.sim-start "output.opal"
21  opal0.sim-step 10000
22  ruby0.dump-stats "output.ruby"
23  opal0.stats
```

Va notato che il tempo di simulazione con l'introduzione dei moduli GEMS si dilata enormemente: vengono eseguiti un miliardo di cicli in circa 10 ore.

3.5 Magic instruction

Dal momento che la simulazione microarchitetturale è enormemente più onerosa rispetto a quella funzionale, occorre trovare il modo per poter passare alla simulazione microarchitetturale solo nel momento in cui viene eseguito il benchmark. Per tale scopo Simics mette a disposizione le *magic instruction*. Queste istruzioni sono trasparenti alla macchina target ma vengono intercettate dal simulatore e gestite tramite appositi handler che possono contenere istruzioni per sospendere la simulazione. Il codice C per generare una magic instruction è il seguente:

```
1  #include<magic-instruction.h>
2
3  void *global_data;
4  int main()
5  {
6    MAGIC(1);
7  }
```

L'header file *magic-instruction.h* viene fornito con Simics. L'istruzione `MAGIC(1)` viene tradotta dipendentemente dall'architettura simulata secondo la tabella 3.3. L'hap handler può essere contenuto in uno script python[1] ed avere la struttura seguente:

```
1  #Creo l'hap per gestire la magic_instruction
2  def magic_fun(user_arg, cpu, userpar):
3          if( userpar ==1):
4                  print "Ora fermo il simulatore"
```

Architettura	Istruzione	Limiti	Compilatori
ia64	nop 0x1?????	0<=n<0x100000	gcc
mips	li $zero,n	0<=n<=0xffff	gcc
ppc	mr n,n	0<=n< 32	gcc
ppc64	fmr n,n	0<=n<32	gcc
sparc	sethi n,%g0	0<n<(1«22)	gcc, WS C[++]
x86	xchg bx,bx	n==0	gcc

Tabella 3.3: Traduzione magic instruction

```
5                    eval_cli_line("sim-break 1")
6   SIM_hap_add_callback("Core_Magic_Instruction", magic_fun, None)
```

La chiamata alla magic instruction può essere effettuata nella macchina target con uno script sh come questo:

```
1   ./magic_instruction
2   ./sqlplus "oracle/oracle" < 1.sql
3   ./magic_instruction
```

La prima magic instruction provocherà il passaggio alla simulazione microarchitetturale mentre la seconda permetterà di fermare Simics per prendere nota delle statistiche della simulazione[14].

3.6 Checkpoint

Simics mette a disposizione dell'utente il meccanismo dei *checkpoint*. Il comando per scrivere un checkpoint è `write-configuration` mentre quello per leggerlo è `read-configuration`. I checkpoint sono delle fotografie istantanee dello stato della macchina target. Viene memorizzato lo stato dei processori, delle memorie, e dei dischi. La comodità di questo sistema è che checkpoint successivi fotografano soltanto le differenze rispetto a quelli precedenti il che significa che è possibile creare un checkpoint ad ogni nodo cruciale dell'installazione, avendo così la possibilità di ritornare sulle scelte fatte conservando tutto il lavoro precedente.

L'altra faccia della medaglia è che i checkpoint occupano molto spazio. La macchina da noi configurata richiede circa 7.4 GB di spazio per le sole immagini dei dischi. A questo, per ogni checkpoint creato, va aggiunto spazio sufficiente per memorizzare tutte le variazioni rispetto allo stato della macchina target. Lo spazio occupato dipende quindi da quante variazioni ci sono state dello spazio di memoria della macchina target rispetto al checkpoint precedente e non è quindi prevedibile se non in senso lato. Per dare un'idea di quali possano essere queste

[14] vedremo nel paragrafo 6.5 come interrompere la simulazione in un momento più opportuno.

dimensioni, creare una decina di checkpoint con variazioni minime ha richiesto uno spazio di altri 8 GB.

Oltre ai checkpoint esiste un altro modo per salvare lo stato dei soli dischi interfacciati alla macchina target; con *write-persistent-data* è possibile salvare lo stato del disco mentre con *read-persistent-data* è possibile richiamarlo. La consistenza del disco deve essere garantita dall'utente che deve quindi usare questi comandi solo dopo aver sganciato il disco dal filesystem.

3.7 Craff

Benché comodissimi, i checkpoint possono crescere troppo di numero e generare confusione. Virtutech distribuisce anche l'utility `craff` che permette di unire insieme più checkpoint. Il risultato di questa fusione è una nuova serie di file indipendenti che rappresentano l'ultimo checkpoint effettuato. Si possono così cancellare i file relativi ai checkpoint che abbiamo unito per recuperare un po' di leggibilità nella gerarchia dei file.

CAPITOLO 4

Solaris

4.1 Presentazione

Solaris è l'implementazione di Sun Microsystems[10] per processori Sparc o Intel x86 del sistema operativo Unix. Le caratteristiche principali di Solaris sono affidabilità e sicurezza, per questo motivo il sistema operativo di Sun è da tempo utilizzato dagli Internet provider e nelle grandi aziende per eseguire le applicazioni più critiche. La nuova versione 10 è disponibile in due varianti: per i grandi server con processore Sparc in tecnologia Risc, e per i normali computer e server che usano Cpu di classe Pentium o superiore.

I cinque Cd-Rom della distribuzione, localizzata anche in lingua italiana, contengono il porting di alcuni pacchetti OpenSource significativi come il web server Apache. Solaris 10 esegue anche programmi Linux in formato binario, grazie al nuovo emulatore Janus integrato nel sistema. L'innovazione più importante di Solaris 10 è il supporto per i container, che sono ambienti di esecuzione isolati tra loro ma capaci di condividere i servizi del sistema operativo, senza la perdita di efficienza causata dalle tecniche di emulazione. Secondo Sun, Solaris 10 è un'alternativa preferibile ai concorrenti Virtual Pc e VMware in ambiente Windows o Linux, quando è necessario consolidare più server per ridurre i costi di acquisto e amministrazione. L'unico limite al numero di container o *server virtuali* è dato dalla memoria e potenza di calcolo del server: il sistema operativo si accontenta di 512 Mbyte di Ram, ma per sfruttare i container ne occorrono almeno il doppio. Anche il filesystem ZFS[1] di Solaris 10 è studiato per i server di rete. Attivandolo, ogni file memorizzato sul disco è automaticamente protetto da un codice crc che segnala, e se possibile corregge, ogni corruzione dei dati provocata da guasti,

[1] Zettabyte File System

errori di configurazione dell'hardware o sbalzi di tensione. Il filesystem ZFS è di tipo journaled, usa 128 bit per indicizzare i settori del disco e consente l'espansione e la riduzione della capacità di qualsiasi ramo del filesystem senza bisogno di eseguire copie, riformattazioni o cambiamenti dei mountpoint: una caratteristica utilissima quando c'è bisogno di aggiungere o rimuovere dischi. Le prestazioni di ZFS sono supportate da un nuovo stack TCP/IP[2], che secondo i dati forniti da Sun è un grande passo avanti rispetto alle possibilità della precedente versione di Solaris. Tutte le caratteristiche del sistema si possono studiare e configurare in dettaglio con l'abbondante documentazione tecnica in lingua inglese disponibile sul sito web di supporto[3].

Solaris ha molte delle funzionalità tipiche dei sistemi real-time[12]. Queste funzionalità sono:

- Un kernel multithreaded preemptable
- Global priority model: threads are mapped to lightweight processes, which are allocated to priority classes and then scheduled globally.
- Configurable clock tick: the frequency of the clock tick can be changed, thereby increasing or decreasing the frequency with which the scheduler runs.
- Timers POSIX ad alta risoluzione: Solaris implementa un POSIX timer addizionale (CLOCK_HIGHRES) che, basato sulle capacità dell'hardware, permette risoluzioni di tempi in nano secondi.
- Priority I/O streams
- Supporto aggiuntivo per le API POSIX real-time.
- Symmetrical multiprocessing support: Solaris supporta i sistemi multiprocessori in modo trasparente all'utente.

4.2 Come Solaris implementa i thread

Solaris implementa sia gli *user-level thread* che i *kernel-level thread*. Gli user-level thread sono implementati come librerie a livello utente mentre i kernel-level thread sono unità di esecuzione viste dal kernel.

Solaris usa il meccanismo degli LWP[3] per eseguire i thread di livello kernel sui processori. Il mapping dei thread user-level agli LWP può essere fatto in vari modi. Se più thread user-level sono mappati su un singolo thread kernel-level, al più uno di questi può essere attivo in un certo momento. Per trarre vantaggio da un

[2]Transmission Control Protocol / Internet Protocol

[3]lightweight processes

sistema multiprocessore i thread user-level possono essere mappati univocamente con altrettanti LWP.

La figura 4.1 mostra come si possono associare processori ai task real-time[4]. Il comando *psrset* è inizialmente usato per creare set di uno o più processori. Va notato che un processore viene riservato dal sistema operativo per l'esecuzione dei processi LWP e non può quindi far parte di alcun set. Il comando *psradm* può essere usato per disabilitare le interruzioni *unbound* nei processori del set. Il comando psrset è quindi usato per associare i processi real-time al set di processori scelto. Tutti gli altri processi ed interrupt possono essere eseguiti al di fuori del set dedicato al real-time.

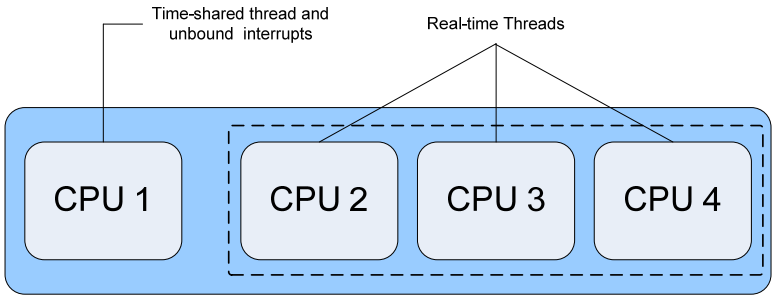

Figura 4.1: Possibile binding per processi real-time

Per supportare differenti politiche di schedulazione, Solaris esegue ogni processo LWP in una delle quattro classi di priorità della tabella 4.1. Le routine interrupt non fanno parte del processo di schedulazione ma vengono eseguite ad una priorità maggiore rispetto a tutti gli altri task.

Class	Priority range	Description
ISRs	N/A	Asynchronous interrupt service routines; not scheduled
Interrupt threads	160-269	Interrupt processing not done in the ISR; scheduled based on priority of ISR
Real-time	100-259	Time-critical tasks; fixed priority preemptive scheduling
System/kernel	60-99	System level functions
Time sharing/Interactive	0-59	General-purpose applications; OS may dynamically adjust priorities to achieve fairness

Tabella 4.1: Solaris priority classes

Quando viene invocato lo schedulatore questo richiama l'LWP con la priorità maggiore, se sono disponibili più CPU vengono eseguiti più LWP.

[4]il termine *real-time* è qui usato nell'accezione comune

Per i thread in timesharing lo schedulatore può incrementare la priorità dell'LWP se ritiene che un thread non stia ricevendo un trattamento equo nell'uso della CPU. Solaris può anche decidere di promuovere un LWP a livello sistema se questo sta controllando una risorsa di sistema.

4.3 Strumenti

Solaris mette a disposizione dell'amministratore di sistema (utente root) alcuni strumenti per il controllo dei processori, dei processi e per il binding dei primi ai secondi. La tabella 4.2 mostra i principali comandi Solaris per l'amministrazione dei processi.

Comando	**Funzione**
cpustat	stampa le statistiche dei processori
pbind	controlla il binding dei processi o degli LWP
pooladm	per attivare e disattivare le *resource pools facility*
prstat	per vedere i processi e gli LWP collegati
psradm	per cambiare lo stato operazionale dei processori
psrinfo	stampa informazioni sui processori
psrset	per definire ed assegnare ai processi *set* di processori
trapstat	stampa le statistiche dei trap

Tabella 4.2: Strumenti Solaris per il controllo dei processori e dei processi

Di seguito una tipica sequenza di messaggi che Solaris mostra al momento del *bootstrap* una macchina Sparc:

```
Switching to high addresses
Setting up TLBs Done
MMU ON
PC = 0000.01ff.f000.1cb8
PC = 0000.0000.0000.1d24
Decompressing into Memory Done
Size = 0000.0000.0006.d840
ttya initialized
Using POST's System Configuration
Setting up memory
Clock board TOD does not match TOD on any IO board.
fhc ac simm-status environment sram flashprom SUNW,UltraSPARC-II
fhc ac simm-status environment sram flashprom
Probing UPA Slot at 2,0   sbus fhc ac environment flashprom eeprom sbus-speed co
unter-timer
Probing UPA Slot at 3,0   sbus counter-timer
Probing /sbus@2,0 at 1,0  Nothing there
Probing /sbus@2,0 at 2,0  Nothing there
Probing /sbus@3,0 at 3,0  SUNW,hme SUNW,fas sd st
Probing /sbus@3,0 at 0,0  Nothing there
16-slot Sun Enterprise E6500, No Keyboard
OpenBoot 3.2.29, 2048 MB memory installed, Serial #8407660.
Copyright 2001 Sun Microsystems, Inc.  All rights reserved
Ethernet address 10:10:10:10:10:12, Host ID: 80804a6c.


Boot device: /sbus@3,0/SUNW,fas@3,8800000/sd@1,0  File and args: -v
Loading ufs-file-system package 1.4 04 Aug 1995 13:02:54.
FCode UFS Reader 1.12 00/07/17 15:48:16.
Loading: /platform/SUNW,Ultra-Enterprise/ufsboot
Loading: /platform/sun4u/ufsboot
The boot filesystem is logging.
```

```
The ufs log is empty and will not be used.
Size: 0x77f70+0x1baaa+0x2fcaa Bytes
module /platform/sun4u/kernel/sparcv9/unix: text at [0x1000000, 0x1077f6f] data at 0x1800000
module misc/sparcv9/krtld: text at [0x1077f70, 0x1090867] data at 0x184b758
module /platform/sun4u/kernel/sparcv9/genunix: text at [0x1090868, 0x11de65f] data at 0x1850f40
module /platform/SUNW,Ultra-Enterprise/kernel/misc/sparcv9/platmod: text at [0x11de660, 0x11dec9f] data at 0x18a2a90
module /platform/sun4u/kernel/cpu/sparcv9/SUNW,UltraSPARC-II: text at [0x11decc0, 0x11eae57] data at 0x18a32c0
SunOS Release 5.10 Version Generic_118822-25 64-bit
Copyright 1983-2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Ethernet address = 10:10:10:10:10:12
mem = 2097152K (0x80000000)
avail mem = 2041077760
root nexus = 16-slot Sun Enterprise E6500
pseudo0 at root
pseudo0 is /pseudo
scsi_vhci0 at root
scsi_vhci0 is /scsi_vhci
sbus1 at root: UPA 0x3 0x0 ...
sbus1 is /sbus@3,0
/sbus@3,0/SUNW,fas@3,8800000 (fas0):
        rev 2.2 FEPS chip
fas0 at sbus1: SBus1 slot 0x3 offset 0x8800000 and slot 0x3 offset 0x8810000 SBus level 3 sparc9 ipl 5
fas0 is /sbus@3,0/SUNW,fas@3,8800000
sd0 at fas0: target 1 lun 0
sd0 is /sbus@3,0/SUNW,fas@3,8800000/sd@1,0
WARNING: Last shutdown is later than time on time-of-day chip; check date.
root on /sbus@3,0/SUNW,fas@3,8800000/sd@1,0:a fstype ufs
fhc0 at root: UPA 0x0 0xf8800000
fhc0 is /fhc@0,f8800000
ac0 board 0 bank 0: base 0x0 size 1024mb rstate 2 ostate 1 condition 1
ac0 board 0 bank 1: base 0x40000000 size 1024mb rstate 2 ostate 1 condition 1
ac0 is /fhc@0,f8800000/ac@0,1000000
fhc3 at root: UPA 0x4 0xf8800000
fhc3 is /fhc@4,f8800000
ac2 is /fhc@4,f8800000/ac@0,1000000
fhc2 at root: UPA 0x2 0xf8800000
fhc2 is /fhc@2,f8800000
ac1 is /fhc@2,f8800000/ac@0,1000000
central0 at root: UPA 0x1f 0x0 ...
fhc1 at root: UPA 0x0 0xf8800000
fhc1 is /central@1f,0/fhc@0,f8800000
sysctrl0 is /central@1f,0/fhc@0,f8800000/clock-board@0,900000
sysctrl0: Key switch is not in the secure position
environ0 is /fhc@0,f8800000/environment@0,400000
environ2 is /fhc@4,f8800000/environment@0,400000
environ1 is /fhc@2,f8800000/environment@0,400000
simmstat0 is /fhc@0,f8800000/simm-status@0,600000
simmstat1 is /fhc@4,f8800000/simm-status@0,600000
sram0 is /fhc@0,f8800000/sram@0,200000
sram1 is /fhc@4,f8800000/sram@0,200000
pseudo-device: dld0
dld0 is /pseudo/dld@0
zs0 is /central@1f,0/fhc@0,f8800000/zs@0,902000
zs1 is /central@1f,0/fhc@0,f8800000/zs@0,904000
cpu0: UltraSPARC-II (portid 0 impl 0x11 ver 0x0 clock 168 MHz)
iscsi0 at root
iscsi0 is /iscsi
SUNW,hme0 : Sbus (Rev Id = 22) Found
hme0 at sbus1: SBus1 slot 0x3 offset 0x8c00000 and slot 0x3 offset 0x8c02000 and slot 0x3 offset 0x8c04000 and slot
0x3 offset 0x8c06000 and slot 0x3 offset 0x8c07000 SBus level 4 sparc9 ipl 7
hme0 is /sbus@3,0/SUNW,hme@3,8c00000
dump on /dev/dsk/c0t1d0s1 size 512 MB
pseudo-device: devinfo0
devinfo0 is /pseudo/devinfo@0
pseudo-device: tod0
tod0 is /pseudo/tod@0
pseudo-device: pm0
pm0 is /pseudo/pm@0
Hostname: peanut
sd2 at fas0: target 2 lun 0
sd2 is /sbus@3,0/SUNW,fas@3,8800000/sd@2,0
checking ufs filesystems
/dev/rdsk/c0t2d0s0: is logging.
/dev/rdsk/c0t1d0s7: is logging.

peanut console login: root
pseudo-device: pool0
pool0 is /pseudo/pool@0
Last login: Tue Jun  4 23:26:44 on console
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
#
```

4.4 Solaris su Simics

Virtutech distribuisce insieme a Simics alcune immagini di dischi preparati per l'avvio di Linux su quasi tutti i sistemi emulati in tabella 3.1. Per motivi di licenza però non vengono distribuiti i disk dump di macchine Sunfire con Solaris installato. È necessario quindi procurarsi Solaris, scaricabile dal sito di Sun e procedere con l'installazione sotto Simics.

Solaris è in grado di riconoscere all'avvio le modifiche alla configurazione della macchina quindi per l'installazione con Simics è sufficiente predisporre una macchina target con un solo processore e una minima quantità di memoria (256 MB). L'installazione di Solaris è interattiva e richiederebbe l'intervento dell'utente. Il processo di installazione può durare fino ad un paio di giorni su una macchina simulata e sostanzialmente obbliga l'utente alla vigilanza continua della console per effettuare le varie scelte di installazione. Fortunatamente Virtutech mette a disposizione script di installazione di Solaris 8, 9 e 10 che rispondono al posto dell'utente durante l'installazione. Simics è in grado di intercettare l'output della macchina target con il comando `<text-console>.wait-for-string` e di rispondere con appositi input tramite il comando `<text-console>.input` dove `<text-console>` va sostituita con la variabile che detiene l'handler della console simulata.

4.5 I demoni di Solaris e gli script di avvio

Il programma di installazione di Solaris predispone vari servizi che possono essere avviati al boot automaticamente oppure tramite i relativi startup-scripts. Alcuni di questi *demoni* possono presentare problematiche di sicurezza, altri sono in molti casi inutili, altri ancora possono generare una quantità di dati e informazioni, o offrire servizi che di fatto non sono utili per i nostri scopi.

Alcuni di questi servizi interagiscono in modo stretto con l'hardware della macchina e generano per tanto istruzioni non supportate dal simulatore microarchitetturale. Questi servizi devono essere spenti prima del passaggio alla simulazione con Ruby + Opal.

Vediamo una rassegna dei servizi che si possono ritrovare come installati ed eseguiti al boot in una installazione standard di Solaris[6].

- **S01MOUNTFSYS**: esegue il mount dei *file system*. Presente in ogni run-level;
- **S05RMTMPFILES**: rimuove file temporanei e ripulisce le relative directory;
- **S20sysetup**: esegue alcuni script di startup;

- **S21perf**: esegue (vanno scommentati) gli script di *System Activity Reporting* (sar) che generano log e statistiche sull'utilizzo delle risorse della macchina. Utile solo per monitoring approfondito;
- **S30sysid.net**: completa la configurazione delle interfacce di rete;
- **S40llc2**: inizializza il *Logical Link Control driver* (Layer 2 networking);
- **S47asppp**: esegue il *aspppd*, *Asyncronous PPP manager*. Utile solo in stazioni con modem per dial in e dial out. Utile solo se si usa un modem;
- **S69inet**: imposta la configurazione del TCP/IP impostando i vari parametri di rete;
- **S70uucp**: esegue operazioni di manutenzione per file gestiti con uucp (antico metodo di copia file fra stazioni Unix). Utile solo se si usa uucp;
- **S71ldap.client**: lancia il client *ldap*. Necessario solo se si usa ldap;
- **S71rpc**: lancia il *rcpbind* (Remote Procedure Call manager) e i servizi NIS+. Potenzialmente pericoloso per la sicurezza. Necessario se si usa NIS+, NFS o altri servizi RPC;
- **S71sysid.sys**: esegue una serie di comandi che compeltano l'inizializzazione del sistema;
- **S72inetsvc**: completa la configurazione della rete. Viene lanciato, se configurato, il server DNS;
- **S72autoinstall**: esegue script di post-installazione dopo un setup via rete (con Kickstart). Necessario solo in fase di installazione con Kickstart;
- **S72slpd**: esegue il *Service Locator Protocol Daemon*, necessario per l'autodiscovery di certi servizi in rete. Necessario solo in ambienti che supportano questo protocollo;
- **S73nfs.client**: esegue i processi necessari per il client NFS;
- **S73cachefs.daemon**: gestisce il *Cache File system*, per migliorare le prestazioni del FS;
- **S74autofs**: avvia *automountd* che permette il mount automatico di device;
- **S74xntpd**: lancia il *Network Time Protocol Daemon*;
- **S74syslog**: esegue il *syslogd*, fondamentale per il logging del sistema;
- **S75cron**: esegue il demone che gestisce il cron;

- **S75savecore**: gestisce i file di *coredump* (core);
- **S76nscd**: lancia il *Name Service Cache Daemon*;
- **S80PRESERVE**: sposta file editati con Ex in /usr/preserve;
- **S80lp**: lancia il *Print Server Daemon*;
- **S80spc**: esegue *printd*, servizio di supporto per *lpd*;
- **S85power**: gestisce il *Power Management*;
- **S88utmpd**: avvia il demone *utmpd*, che monitora il corretto accounting dei processi su /var/adm/utmp;
- **S89bdconfig**: lancia *bdconfig* per la gestione di Buttons e Dials;
- **S90wbem**: necessario per il *Web Based Enterprise Management*;
- **S92volmgt**: lancia il *Volume Management Daemon*;
- **S93cacheos.finish**: esegue script utile per il *cacheos*;
- **S94ncalogd**: gestisce il logging del *Network Cache And Accelerator Server*;
- **S95ncad**: esegue il *Network Cache And Accelerator server*, utile per migliorare le prestazioni di un server web;
- **S99audit**: gestisce l'*audit daemon*;
- **S99dtlogin**: lancia il *dtlogin* per il login su ambiente grafico CDE;

Al termine del boot di Solaris sono stati rimossi alcuni servizi usando gli script di inizializzazione in modo da avere meno interferenze possibili durante le esecuzioni delle query.

webconsole	volmgt	ufs_quota
sysetup	swupboots	slpd
sendmail	samba	sncd
nfs.server	mkdtab	lu
drvconfig	devlinks	deallocate
audit	appserv	apache
acctadm		

Tabella 4.3: Script di startup/shutdown dei servizi Solaris

Gli script per lo startup/shutdown dei servizi sono riportati in tabella 4.3, per fermare l'esecuzione di detti servizi basta digitare (prendendo ad esempio il primo):

```
# /etc/init.d/webconsole stop
```

Solaris risponde con un messaggio che conferma l'avvenuto startup o shutdown del servizio.

CAPITOLO 5

Oracle

5.1 Presentazione

Oracle è il RDBMS[1] più diffuso al mondo per la gestione dei dati sui server Unix e Windows. Permette la parallelizzazione delle query SQL[2] ed incorpora una serie di caratteristiche che lo hanno reso popolarissimo negli ambienti datawarehouse:

- Collegabilità
  - Oracle permette di disporre dei dati residenti su sistemi incompatibili tra loro, come se fossero tutti sullo stesso sistema;
  - le limitazioni connesse con la diversità dell'hardware e dei sistemi operativi sono virtualmente eliminate, infatti Oracle permette alle applicazioni di accedere ai dati di un qualunque server Oracle in rete, collegante differenti ambienti;
  - il database administrator ha la possibilità di selezionare l'hardware più opportuno per eseguire una particolare funzione essendo garantita la compatibilità del software;
  - il carico delle elaborazioni può essere distribuito più efficientemente;
  - i meccanismi di sicurezza e di recovery si estendono su tutta la rete;
  - sono supportati i protocolli di comunicazioni più diffusi;
  - è prevista la concorrenza al fine di assicurare l'integrità dei dati degli ambienti multiutente:

[1] Relational Database Management System
[2] Sequel Language

    - accesso trasparente dell'utente a tutti i suoi dati;
    - memorizzati in differenti DBMS;
    - gestiti con differenti computer;
    - attraverso differenti sistemi operativi come se fossero sulla stessa macchina;
  - sono supportati tutti i DBMS realmente SQL;

- Produttività
  - Permette la diminuzione sia del Back Log del software applicativo che degli sforzi per mantenere quello sviluppato;
  - Mette a disposizione una tecnologia software utilizzabile sia dagli specialisti che dagli utenti finali;
  - Offre interfacce user-friendly tipo Windows;
  - Permette la scrittura di programmi in linguaggi tradizionali;
  - Dispone di un'interfaccia interattiva al DBMS ORACLE per utenti che hanno scarsa familiarità con il database management system e con il linguaggio SQL;
  - Dispone di uno strumento per l'analisi e la formulazione delle previsioni;

- Transazionabilità
  - Permette di disporre di una *software database machine* per gestire grandi basi di dati, un elevato volume di transazioni ed essere in grado di assicurare un funzionamento continuo;
  - *On Line Transaction Processing*;
  - combina la caratteristica della piena trasportabilità delle applicazioni con l'elevato throughput richiesto dalle applicazioni OLTP usando architetture che superano i colli di bottiglia della CPU e delle I/O;
  - utilizza in pieno le architetture multiprocessing;
  - alta concorrenzialità;
  - lock a livello di riga di dati;
  - elevata disponibilità e fault tolerance;
  - supporta i DBMS di grandi dimensioni;
  - dispone di un linguaggio procedurale integrato con il database.

5.2 Installazione

Oracle è disponibile sul sito `www.oracle.com` compilato per diverse piattaforme. La versione per Solaris, una volta scaricata e scompattata con il comando:

```
# cpio -idmv < 10gr2_db_sol.cpio.gz
```

si installa mediante lo script di installazione `runInstaller`. Il comando esegue l'utility di installazione che gira in ambiente grafico. Oracle prevede anche un *silent mode* di installazione mediante *response files* nei quali vanno specificate le opzioni di installazione e la cui messa a punto richiede tempo giustificabile solo in mancanza della disponibilità del server grafico o qualora l'installazione vada reiterata molte volte.

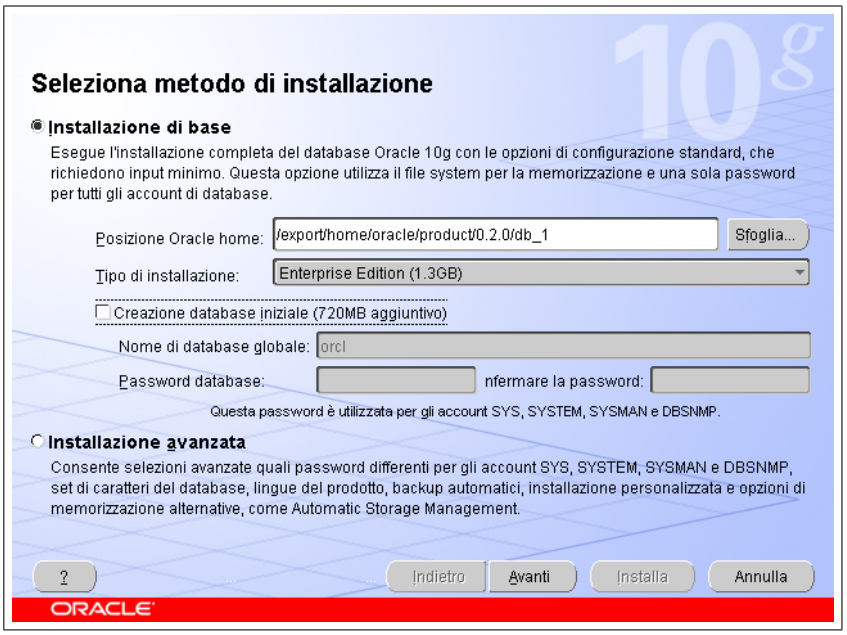

Figura 5.1: Il programma di installazione di Oracle

Salvo diversamente specificato nelle opzioni di `runInstaller`, al termine dell'installazione viene lanciato il *Database Configurator Assistant* che guida l'utente nella configurazione di un database.

Per ovviare ai problemi di lentezza delle operazioni si è preferito installare il solo software senza creare la base di dati e successivamente importare quest'ultima mediante le utility `exp` ed `imp` di Oracle. L'importazione delle tabelle TPC-H sul database Oracle è stata demandata ad un host reale. Oracle è stato quindi installato parallelamente anche su una macchina Windows dual-core sulla quale è stato creato e popolato il database secondo i vincoli imposti da TPC[3]-H. Suc-

[3]Transaction Processing Performance Council

cessivamente il database popolato, l'utente `oracle` e i relativi permessi sono stati esportati nel file `EXPDAT.DMP` e importati successivamente nella macchina target.

I compiti sono stati ripartiti tra le macchine come in tabella 5.1. La macchina 'sun' è una macchina reale (vedi tabella A.1 ed è stata utilizzata per mettere a punto le procedure di configurazione in ambiente Solaris/Oracle data la sua immediata risposta rispetto alla macchina simulata.

Macchina	**Compiti**
vega (host)	creazione tabelle TPC-H
chomp	conversione tabelle in database Oracle configurazione permessi esportazione database Oracle
peanuts (vega)	importazione database Oracle
sun	messa a punto procedure

Tabella 5.1: Compiti assegnati ai sistemi disponibili

5.3 Configurazione

5.3.1 Solaris

È necessario apportare qualche modifica alla configurazione di Solaris. Il file di sistema `/etc/system` deve includere i seguenti settaggi, il cui significato è riportato in tabella 5.2:

```
set shmsys:shminfo_shmmax=4294967295
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmseg=10
set semsys:seminfo_semmns=1024
set semsys:seminfo_semmsl=256
set semsys:seminfo_semmni=100
set semsys:seminfo_semvmx=32767
set noexec_user_stack=1
```

È utile anche definire un nuovo utente `oracle` per permettere l'installazione del DBMS. Infatti Oracle non permette l'installazione all'utente `root`. In ambiente Solaris un utente può essere creato con il comando

```
# useradd -d /export/home/oracle -m -s /bin/sh oracle
```

successivamente può essere impostata la sua password da `root` con il comando

```
# passwd oracle
```

Parametro	Descrizione
SHMMAX	Massima dim del segmento di memoria condivisa
SHMMIN	Min dim, di cui sopra
SHMMNI	Massima dimensione del shared memory identifiers
SHMSEG	Max num di shared memory segment per processo utente
SEMMNI	Max numero di semaphore identifiers
SEMMSL	Max numero di semafori in un set
SEMMNS	Max numero di semafori nel sistema
SEMOPM	Max numero di operazioni per semop call
SEMVMX	Max numero di semafori

Tabella 5.2: Parametri kernel

È buona precauzione settare correttamente la data del sistema target mediante

```
# Date -u 020212122006
```

che imposta come data di sistema il 2 febbraio 2006, ore 12:12. È stato riscontrato infatti che il programma di installazione di Oracle si rifiuta di installare il DBMS nel caso in cui la data del sistema sia precedente a quella della creazione del pacchetto di installazione.

5.3.2 L'interfaccia HTML

Oracle dispone di un'interfaccia HTML[4] che risulta molto comoda per la configurazione del database. Per attivare questa interfaccia occorre che il database sia istanziato, si attiva quindi il *listener*:

```
$ ./lsnrctl start
```

e quindi la console HTML:

```
$ ./emctl start dbconsole
```

A questo punto è possibile connettersi alla console tramite la porta 1158 all'indirizzo http://localhost:1158/em (nel caso in cui ci si connetta dalla stessa macchina su cui è stata avviata l'istanza Oracle) e dopo il *login*, accedere alla pagina di figura 5.2. L'interfaccia HTML permette di effettuare alcune impostazioni senza che sia necessaria alcuna conoscenza di SQL e fornisce, inoltre, grafici statistici circa l'occupazione del database in memoria, la sua indicizzazione e informazioni circa gli accessi ai dati.

[4]HyperText Markup Language

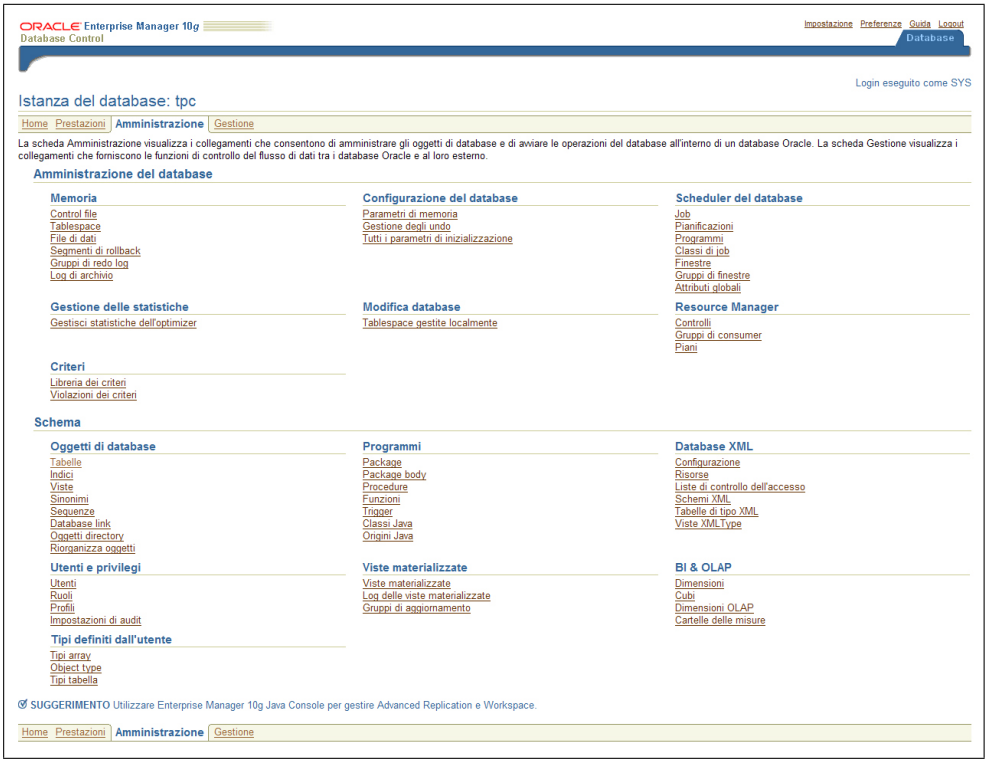

Figura 5.2: Pagina di amministrazione di Oracle

5.3.3 Oracle

È stato creato l'utente `oracle` al quale sono stati assegnati i privilegi di `CONNECT`, `DBA`, `EXP_FULL_DATABASE` e `IMP_FULL_DATABASE` mediante il comando SQL:

```
CREATE USER "ORACLE" PROFILE "DEFAULT" IDENTIFIED BY "oracle"
  DEFAULT TABLESPACE "USERS"
  TEMPORARY TABLESPACE "TEMP"
  ACCOUNT UNLOCK
  GRANT "CONNECT" TO "ORACLE"
  GRANT "DBA" TO "ORACLE"
  GRANT "EXP_FULL_DATABASE" TO "ORACLE"
  GRANT "IMP_FULL_DATABASE" TO "ORACLE";
```

I privilegi EXP_FULL_DATABASE e IMP_FULL_DATABASE permettono a chi li detiene di effettuare l'importazione o l'esportazione del database dell'utente mediante le utility di Oracle. CONNECT è il privilegio di default assegnato ai nuovi utenti; DBA garantisce all'utente il privilegio di amministrare il database.

Per il momento dichiariamo parallele con grado di parallelismo di default (dinamicamente calcolato da Oracle sulla base dei processori disponibili) le tabelle costituenti il database.

```
ALTER TABLE lineitem CACHE PARALLEL;
ALTER TABLE supplier CACHE PARALLEL;
ALTER TABLE region CACHE PARALLEL;
ALTER TABLE nation CACHE PARALLEL;
ALTER TABLE part CACHE PARALLEL;
ALTER TABLE partsupp CACHE PARALLEL;
ALTER TABLE orders CACHE PARALLEL;
ALTER TABLE customer CACHE PARALLEL;
```

Questa scelta è motivata nel paragrafo 6.4.4.

Le dimensioni delle strutture di dati di Oracle sono state impostate come in tabella 5.3.

Area	**Dimensione**
Total System Global Area	612368384
Fixed Size	1980584
Variable Size	171968344
Database Buffers	432013312
Redo Buffers	6406144

Tabella 5.3: Dimensioni strutture dati di Oracle

5.4 System Global Area

Quando si avvia un'istanza di database, Oracle alloca un'area di memoria chiamata SGA[5][7], condivisa da tutti gli utenti del database (figura 5.3). La SGA è suddivisa in varie aree delle quali le due più grandi sono la *database buffer cache* e lo *shared pool*. Le dimensioni di queste aree sono configurabili tramite i parametri contenuti nel file di inizializzazione del database.

La database buffer cache è un'area della SGA utilizzata per contenere i blocchi di dati che vengono letti partendo dai segmenti di dati del database, come tabelle, indici e cluster. La dimensione della database buffer cache è determinata dal parametro DB_CACHE_SIZE situato nel file dei parametri di inizializzazione per il database. La dimensione predefinita per i blocchi di database è impostata tramite il parametro DB_BLOCK_SIZE che viene specificato nel file dei parametri durante la creazione del database. La gestione della dimensione della database buffer cache è una parte importante della gestione e della messa a punto del database.

Il database ha una dimensione di blocco predefinita, tuttavia si possono stabilire aree della cache per dimensioni di blocchi di database differenti e poi creare delle tablespace per utilizzare proprio queste cache.

[5]System Global Area

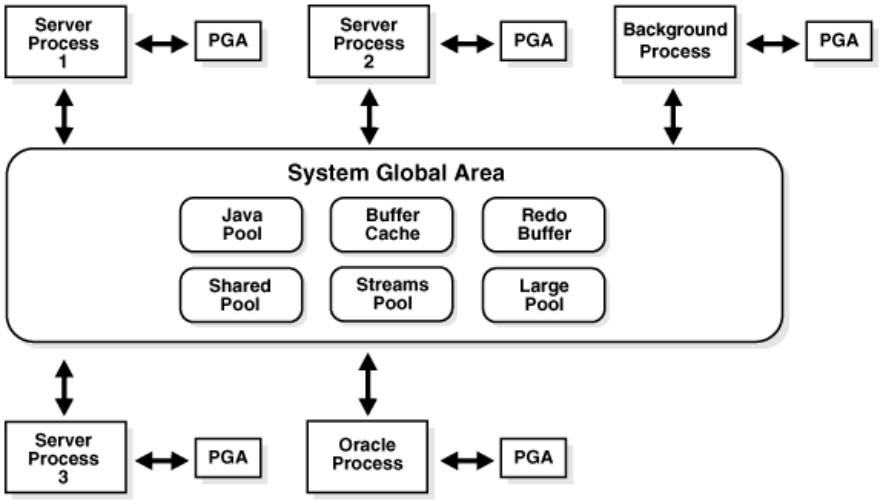

Figura 5.3: La struttura della System Global Area

Generalmente la buffer cache del blocco di dati occupa circa dall'uno al due per cento della dimensione allocata per il database. Oracle gestirà lo spazio disponibile tramite un algoritmo che mantiene il più a lungo possibile nella memoria i blocchi utilizzati con maggior frequenza. Quando nella cache è necessario dello spazio libero, i nuovi blocchi di dati utilizzeranno lo spazio occupato da blocchi cui si accede raramente oppure quello occupato da un blocco modificato dopo che è stato copiato sul disco.

Se la SGA non è abbastanza grande per contenere i dati più utilizzati, oggetti diversi si contenderanno lo spazio all'interno della buffer cache del blocco di dati. E più probabile che questa situazione si verifichi quando più applicazioni usano il medesimo database e quindi condividono la stessa SGA. In questo caso, le tabelle e gli indici utilizzati con maggiore frequenza da ciascuna applicazione si contendono costantemente lo spazio nella SGA con gli oggetti utilizzati più di recente presi dalle altre applicazioni. Di conseguenza, le richieste di dati dalla buffer cache del blocco di dati comporteranno un rapporto inferiore tra "accessi riusciti" e "accessi non riusciti". La buffer cache del blocco di dati manca il risultato negli I/O fisici per quanto riguarda le letture dei dati, portando così ad un calo delle prestazioni. Lo shared pool memorizza la cache del dizionario dati (informazioni relative alle strutture di database) e la *library cache* (informazioni su istruzioni che vengono eseguite sul database). Mentre la buffer cache del blocco di dati e la cache del dizionario dati abilitano la condivisione di informazioni strutturali e di informazioni sui dati tra gli utenti nel database, la library cache consente la condivisione di istruzioni SQL utilizzate comunemente.

Lo shared pool contiene il piano di esecuzione e la struttura di analisi per le istruzioni SQL eseguite sul database. Quando un utente qualsiasi esegue per la seconda volta la medesima istruzione SQL, Oracle potrà sfruttare le informazioni di analisi disponibili nello shared pool per velocizzare l'esecuzione dell'istruzione. Come accade per la buffer cache del blocco di dati, anche lo shared pool è gestito tramite un algoritmo LRU. Man mano che lo Shared Pool si riempie, i percorsi di esecuzione e le strutture di analisi utilizzati meno recentemente verranno rimossi dalla library cache per far spazio alle nuove voci. Se lo shared pool è troppo piccolo, le istruzioni verranno ricaricate continuamente nella library cache, influendo così sulle prestazioni.

La dimensione (in byte) dello shared pool viene impostata tramite il parametro di inizializzazione SHARED_POOL_SIZE. Come per le altre cache, anche la dimensione di Shared Pool può essere alterata mentre il database è aperto.

Le caratteristiche dell'istanza di database, come la dimensione della SGA e il numero dei processi in background, vengono specificate durante la fase di avvio. Questi parametri vengono poi memorizzati in un file binario che prende il nome di file dei parametri di sistema. Per generare file dei parametri di sistema per un database aperto, si usa il comando `create spfile`. Con il comando `create pfile from spfile`, è possibile creare una versione leggibile del file dei parametri di sistema.

CAPITOLO 6

TPC Benchmark™ H

6.1 Presentazione

Il TPC-H[5] è un benchmark di supporto alle decisioni proposto da TPC[4]. È costituito da una base di dati e da una serie di query che ripropongono la situazione di un datawarehouse.

Questo benchmark simula un sistema di supporto alle decisioni caratterizzato da:

- database popolato da grande volume di dati;
- esecuzione di query con alto grado di parallelizzazione;
- risposte a questioni di rilevanza critica nelle scelte aziendali.

Le query TPC-H hanno le seguenti caratteristiche:

- danno risposte a questioni aziendali reali;
- simulano query generate ad-hoc;
- sono più complesse della maggiorparte delle transazioni OLTP[1];
- includono molti operatori e vincoli selettivi;
- generano attività intensive al sottosistema DBMS e al server in generale;
- sono eseguite su un database che verifica certi vincoli di scalabilità;

[1] On Line Transaction Processing

- sono implementate con vincoli derivati dalla realtà.

Ogni benchmark proposto dal consorzio TPC viene definito con un documento di specifiche molto ampio e completo che definisce ogni aspetto (scala o dimensione della base dati, proprietà ACID[2], durata minima dei run di test, popolamento base dati, ecc.). Per essere pubblicati, i risultati dei benchmark debbono essere validati da un Advisor e accompagnati da un documento che riporta tutti i dettagli della configurazione e delle prove svolte. L'impegno per effettuare un benchmark ufficiale è molto oneroso e viene sostenuto solo dai principali vendor hardware e software a fronte del rilascio di nuovi e significativi prodotti. I benchmark TPC sono molto utili poiché forniscono una metrica significativa e di facile raffronto (eg. tpmH: transazioni TPC-H al minuto; Price/tpmH: costo per transazione) e possono essere implementati anche senza soddisfare tutti i requisiti per l'omologazione fornendo comunque risultati significativi. In figura 6.1 è riportato lo schema logico delle tabelle costituenti la base di dati.

6.2 Configurazione

Il benchmark viene distribuito sotto forma di sorgenti in C che vanno compilati adattando il *Makefile* al sistema su cui deve girare. Nel nostro caso i parametri scelti sono quelli di un sistema Solaris con DBMS Oracle:

```
CC       = gcc-2.95
# Current values for DATABASE are: INFORMIX, DB2, TDAT
#                                  SQLSERVER, SYBASE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS
# Current values for WORKLOAD are:  TPCH, TPCR
DATABASE = DB2
MACHINE  = Solaris/SUN
WORKLOAD = TPCH
```

Tpc-h non presenta l'opzione "Oracle" come database ma tale opzione è inutile in quanto Oracle è compatibile, per sintassi delle query, con DB2.

La compilazione produce due eseguibili: dbgen e qgen che rispettivamente popolano il database e generano le query in accordo con la sintassi del DBMS selezionata nel *Makefile*.

6.2.1 dbgen

Il database viene popolato mediante l'utility `qgen` che presenta il seguente output una volta chiamata con il parametro `-h`:

[2]Nell'ambito dei database, ACID deriva dall'acronimo inglese Atomicity, Consistency, Isolation, e Durability (Atomicità, Consistenza, Isolamento e Durabilità)

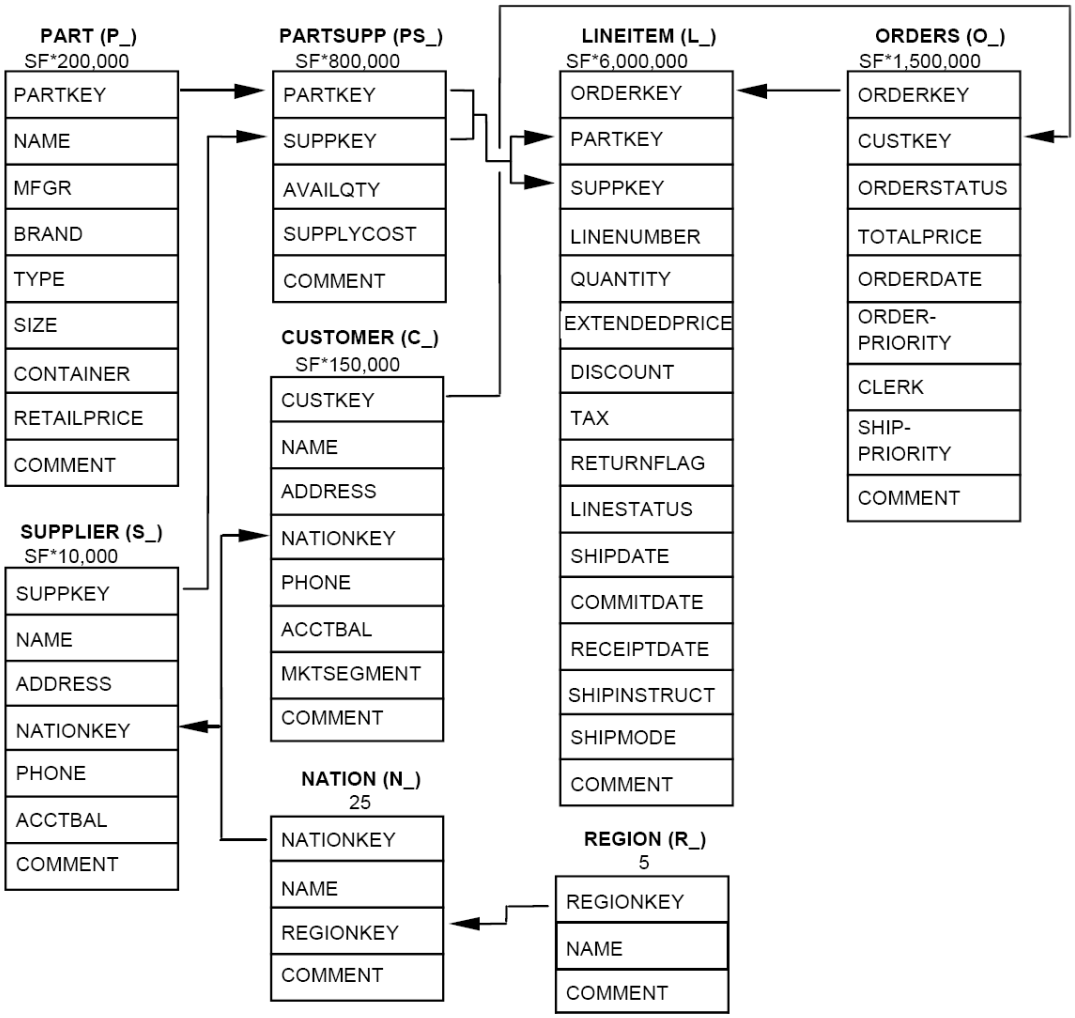

Figura 6.1: Lo schema TPC-H

```
TPC-H Population Generator (Version 2.3.0 )
Copyright Transaction Processing Performance Council 1994 - 2005
USAGE:
dbgen [-{vfFD}] [-O {fhmsv}][-T {pcsoPSOL}]
        [-s <scale>][-C <procs>][-S <step>]
dbgen [-v] [-O {dfhmr}] [-s <scale>] [-U <updates>] [-r <percent>]

-b <s> -- load distributions for <s>
-C <n> -- use <n> processes to generate data
          [Under DOS, must be used with -S]
-D     -- do database load in line
-d <n> -- split deletes between <n> files
-f     -- force. Overwrite existing files
-F     -- generate flat files output
-h     -- display this message
-i <n> -- split inserts between <n> files
-n <s> -- inline load into database <s>
-O d   -- generate SQL syntax for deletes
-O f   -- over-ride default output file names
-O h   -- output files with headers
-O m   -- produce columnar output
-O r   -- generate key ranges for deletes.
-O v   -- Verify data set without generating it.
-q     -- enable QUIET mode
-r <n> -- updates refresh (n/100)% of the
          data set
-s <n> -- set Scale Factor (SF) to  <n>
-S <n> -- build the <n>th step of the data/update set
-T c   -- generate cutomers ONLY
-T l   -- generate nation/region ONLY
-T L   -- generate lineitem ONLY
-T n   -- generate nation ONLY
-T o   -- generate orders/lineitem ONLY
-T O   -- generate orders ONLY
-T p   -- generate parts/partsupp ONLY
-T P   -- generate parts ONLY
-T r   -- generate region ONLY
-T s   -- generate suppliers ONLY
-T S   -- generate partsupp ONLY
-U <s> -- generate <s> update sets
-v     -- enable VERBOSE mode

To generate the SF=1 (1GB), validation database population, use:
        dbgen -vfF -s 1

To generate updates for a SF=1 (1GB), use:
        dbgen -v -U 1 -s 1
```

Le opzioni che a noi interessano sono quelle che regolano il fattore di scala della base di dati. La base di dati standard per i benchmark TPC-H è di 1 GB. L'one-

rosità della simulazione microarchitetturale impone di tenere più basso possibile il fattore di scala.

6.2.2 qgen

Questa utility permette di tradurre le query TPC-H nel dialetto SQL usato dal DBMS. Anche `qgen -h` dispone di una sua pletora di opzioni che riportiamo di seguito:

```
TPC-H Parameter Substitution (v. 2.3.0 )
Copyright Transaction Processing Performance Council 1994 - 2005
USAGE: ./qgen <options> [ queries ]
Options:
-a         -- use ANSI semantics.
-b <str>   -- load distributions from <str>
-c         -- retain comments found in template.
-d         -- use default substitution values.
-h         -- print this usage summary.
-i <str>   -- use the contents of file <str> to begin a query.
-l <str>   -- log parameters to <str>.
-n <str>   -- connect to database <str>.
-N         -- use default rowcounts and ignore :n directive.
-o <str>   -- set the output file base path to <str>.
-p <n>     -- use the query permutation for stream <n>
-r <n>     -- seed the random number generator with <n>
-s <n>     -- base substitutions on an SF of <n>
-v         -- verbose.
-t <str>   -- use the contents of file <str> to complete a query
-x         -- enable SET EXPLAIN in each query.
```

Per i nostri scopi è sufficiente eseguire

```
# qgen $n.sql
```

sostituendo al posto di `$n` il numero della query TPC-H da tradurre. L'output del programma `qgen` verrà passato come input al parser di query di Oracle.

6.2.3 Popolazione del database

Per i problemi elencati precedentemente il database è stato popolato con un fattore di scala 0.1 corrispondente ad una base di dati di 100 MB. Con questa dimensione il database è sufficientemente piccolo da entrare in memoria ram.

La dimensione dettagliata del database è quella riportata in tabella 6.1.

Nome Tabella	N. righe	Dimensione (MB)
SUPPLIER	1 000	0.2
PART	20 000	3
PARTSUPP	80 000	11
CUSTOMER	15 000	2.6
ORDERS	150 000	14.9
LINEITEM	600 572	64.1
NATION	25	<1
REGION	5	<1
TOTAL		**95.6**

Tabella 6.1: Tabelle TPC-H

6.3 Scelta delle query

Il benchmark TPC-H definisce 22 query che devono essere tutte implementate ed eseguite come parte del benchmark. Le query sono divise in 4 parti chiamate *business question*, *functional definition*, *substitution parameters* e *query validation*.

Le *business question* sono modellate su domande reali che il proprietario di un business valuta per prendere decisioni. Le informazioni restituite da queste query includono indicazioni sul pricing, informazioni sulla catena materie prime/fornitori, sulla soddisfazione dei clienti, sui migliori acquirenti, sulla gestione dell'inventario, ecc.

I nostri scopi non comprendono la validazione del benchmark su una macchina simulata per cui possiamo notevolmente sfoltire il numero delle query da considerare e sceglierle per le caratteristiche di esecuzione e di accessi al disco.

La scelta è caduta sulle 5 query di tabella 6.2. Le query scelte sono di sola lettura e sono state selezionate perché già usate in precedenti lavori di ricerca[9].

La query 1.sql è quella mostrata di seguito:

```
1   select
2     l_returnflag,
3     l_linestatus,
4     sum(l_quantity) as sum_qty,
5     sum(l_extendedprice) as sum_base_price,
6     sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
7     sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
8     avg(l_quantity) as avg_qty,
9     avg(l_extendedprice) as avg_price,
10    avg(l_discount) as avg_disc,
11    count(*) as count_order
12  from
```

Query	**Descrizione TPC-H**	**Attività intensiva**	**Tabelle coinvolte**
Q1	Pricing summary report query	full table scan	LINEITEM
Q3	Shipping priority query	hash-join e nested loop join	CUSTOMER ORDERS LINEITEM
Q6	Forecast revenue change query	sort	LINEITEM
Q8	National market share query	hash-join e nested loop join	PART SUPPLIER LINEITEM ORDERS CUSTOMER NATION REGION
Q19	Discounted revenue query	nested loop join	LINEITEM PART

Tabella 6.2: Principali caratteristiche di alcune query TPC-H

```
13     lineitem
14   where
15     l_shipdate <= date '1998-12-01' - interval '[DELTA]' day (3)
16   group by
17     l_returnflag,
18     l_linestatus
19   order by
20     l_returnflag,
21     l_linestatus;
```

6.4 Ottimizzazione su Oracle

Lo scopo principale è di bilanciare il più possibile il carico sui processori in modo da rendere il benchmark il più scalabile possibile all'aumentare del numero di processori. Per fare questo bisogna premettere come Oracle si comporta nella parallelizzazione delle query SQL.

6.4.1 Come Oracle parallelizza le query

Oracle può eseguire le query in modo parallelo. Il grado di parallelismo è definito dal *query coordinator* che, come riportato nei manuali di Oracle:

- analizza la query e determina il DOP[3]
- alloca uno o due set di slaves (thread o processi)
- controlla la query e manda le istruzioni ai PQ[4] slaves
- determina quali tabelle o indici devono essere scandite dai vari PQ
- produce l'output finale per l'utente

Il parallel execution coordinator può arruolare due o più istanze dei parallel execution server per eseguire una query SQL. Il numero di parallel execution server associati con una singola operazione si chiama DOP. Per singola operazione si intende una parte di uno statement SQL come per esempio un *order by* oppure l'esecuzione di un full table scan per eseguire un join. Il DOP si applica al parallelismo *intra-operation*, quello cioè interno al parallel server. Se il parallelismo *inter-operation* è possibile il numero totale di parallel execution server può essere il doppio del DOP.

Non si possono avere contemporaneamente in esecuzione più di due set di parallel execution server. Ogni parallel execution server può processare diverse operazioni. Solo due set di parallel execution server devono essere attivi per garantire il parallelismo ottimale delle *inter-operation*.

L'esecuzione parallela di Oracle è studiata per usare efficientemente CPU e dischi per rispondere rapidamente alle query ma è facile saturare il sistema quando troppi utenti eseguono query parallele. Per ovviare a ciò Oracle implementa diversi modi per controllare il parallelismo:

- un algoritmo adattivo, abilitato per default, che riduce il grado di parallelismo all'aumentare del carico di lavoro sulla macchina
- profiling degli utenti
- il Database Resource Manager che permette il profiling dei gruppi utenti

All'avvio di un'istanza Oracle viene creato un *pool* di parallel execution server. Il numero massimo di processi creati e il numero minimo che Oracle lascia a disposizione è controllabile mediante i parametri di configurazione riportati in tabella 6.3. Il parametro *PARALLEL_MIN_SERVERS* permette di controllare il numero di processi creati all'avvio di un'istanza mentre il parametro *PARALLEL_MAX_SERVERS* specifica il limite massimo del numero di server. Oracle può eseguire query anche in mancanza del numero richiesto di parallel server, nel caso limite passa all'esecuzione seriale. È possibile definire mediante *PARALLEL_MIN_PERCENT* la percentuale di risorse richieste da una query, in

[3]Degree of Parallelism
[4]Parallel Query

Funzione	**Nome Parametro Oracle**
Massimo numero di parallel server	PARALLEL_MAX_SERVERS
Minimo numero di parallel server	PARALLEL_MIN_SERVERS
Percentuale di risorse minime perché venga eseguita una operazione	PARALLEL_MIN_PERCENT
Riduce il DOP di una query basandosi sul carico del sistema al momento dell'inizio dell'esecuzione	PARALLEL_ADAPTIVE_MULTI_USER
Specifica la dimensione dello spazio PGA disponibile ai processi connessi all'istanza di database	PGA_AGGREGATE_TARGET

Tabella 6.3: Parametri di Oracle

termini di numero di parallel server disponibili, al di sotto della quale Oracle non esegue le richieste.

I server vengono consumati dagli utilizzatori (le query SQL) e vengono restituiti al pool una volta liberati. Oracle può decidere di incrementare il numero di parallel server. Quando viene eseguita una query Oracle effettua queste operazioni:

1. il processo che esegue la query diventa un QC[5];
2. il QC ottiene un numero DOP di parallel execution server (se non ce ne sono abbastanza nel pool vengono creati);
3. Oracle esegue gli statement SQL come sequenza di operazioni dove ogni operazione viene, se possibile, eseguita in parallelo;
4. quando uno statement ha finito di essere eseguito, il coordinatore restituisce l'output al processo utente che ha istanziato la query e restituisce il parallel server al pool.

6.4.2 Execution plan

Per pianificare le ottimizzazioni è utile conoscere il *percorso di esecuzione* (o *execution plan*) delle query da eseguire. È possibile generare l'*execution plan* di una query anteponendo al codice SQL il costrutto *explain plan for*.

[5]Query Coordinator

Il percorso di esecuzione può cambiare al variare dei parametri di parallelismo delle tabelle. Supponiamo ad esempio di avere questa semplice query:

```
1  select * from
2          customer,
3          orders
4  where
5          c_custkey = o_custkey
```

Questa query effettua un semplice join delle tabelle `CUSTOMER` e `ORDERS`. Se abbiamo dichiarato entrambe le tabelle coinvolte come non parallele, specificando l'attributo `NOPARALLEL`, otterremo questo piano di esecuzione:

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |  1505K|   476M|       | 20312   (1)| 00:04:04 |
|*  1 |  HASH JOIN         |          |  1505K|   476M|    28M| 20312   (1)| 00:04:04 |
|   2 |   TABLE ACCESS FULL| CUSTOMER |   150K|    27M|       |   901   (1)| 00:00:11 |
|*  3 |   TABLE ACCESS FULL| ORDERS   |  1500K|   205M|       |  6893   (1)| 00:01:23 |
-------------------------------------------------------------------------------------
* access("C_CUSTKEY"="O_CUSTKEY")\newline
* filter("O_CUSTKEY">=0)\newline
```

Tabella 6.4: Execution plan della query con `CUSTOMER` e `ORDERS` dichiarate non parallele

L'esecuzione è seriale: viene prima eseguito un full scan sulla tabella `ORDERS` escludendo le righe che non soddisfano la condizione o_custkey>=0 seguito da un full scan della tabella `CUSTOMER`. Il join viene fatto alla fine sui campi `C_CUSTKEY` e `O_CUSTKEY`.

Più complesso è il caso in cui si dichiara la tabella `CUSTOMER` come parallela:

```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |  1505K|   476M|  7400   (1)| 00:01:29 |        |      |            |
|   1 |  PX COORDINATOR        |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)  | :TQ10002 |  1505K|   476M|  7400   (1)| 00:01:29 |  Q1,02 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN           |          |  1505K|   476M|  7400   (1)| 00:01:29 |  Q1,02 | PCWP |            |
|   4 |     PX RECEIVE         |          |   150K|    27M|   499   (1)| 00:00:06 |  Q1,02 | PCWP |            |
|   5 |      PX SEND HASH      | :TQ10001 |   150K|    27M|   499   (1)| 00:00:06 |  Q1,01 | P->P | HASH       |
|   6 |       PX BLOCK ITERATOR|          |   150K|    27M|   499   (1)| 00:00:06 |  Q1,01 | PCWC |            |
|   7 |        TABLE ACCESS FULL| CUSTOMER |   150K|    27M|   499   (1)| 00:00:06 |  Q1,01 | PCWP |            |
|   8 |     BUFFER SORT        |          |       |       |            |          |  Q1,02 | PCWC |            |
|   9 |      PX RECEIVE        |          |  1500K|   205M|  6893   (1)| 00:01:23 |  Q1,02 | PCWP |            |
|  10 |       PX SEND HASH     | :TQ10000 |  1500K|   205M|  6893   (1)| 00:01:23 |        | S->P | HASH       |
|* 11 |        TABLE ACCESS FULL| ORDERS   |  1500K|   205M|  6893   (1)| 00:01:23 |        |      |            |
-----------------------------------------------------------------------------------------------------------------
* access("C_CUSTKEY"="O_CUSTKEY")
* filter("O_CUSTKEY">=0)
```

Tabella 6.5: Execution plan della query con la sola tabella `CUSTOMER` dichiarata parallela

Viene effettuato un full scan della tabella `CUSTOMER` (id 7) suddividendola in blocchi processati in parallelo dai parallel execution server. Successivamente viene creato un indice hash che viene ricevuto da un *query coordinator*. La tabella

Campo	**Descrizione**
Id	Identificativo dell'operazione
Operation	Operazione
Name	Nome della tabella acceduta o dell'operazione svolta
Rows	Numero di righe coinvolte
Bytes	Spazio in tabella temporanea occupato dall'operazione
Cost	Costo stimato
Time	Tempo stimato
TQ Table queue	Il formato è Qn,m dove n è l'indice del flusso di dati e m è la coda delle tabelle all'interno del flusso di dati. Le query parallele spesso hanno un solo flusso di dati quindi il formato sarà Q1,m
IN-OUT	`S->P`: esecuzione seriale. L'output viene partizionato o inviato in broadcast ai server di esecuzione parallela `P->S`: esecutione parallela. L'output viene inviato ad un query coordinator seriale `P->P`: esecuzione parallela con output ripartizionato ad un secondo set di server di esecuzione paralleli `PWP`: esecuzione parallela. L'output viene processato dallo stesso processo parallelo. Non ci sono comunicazioni fra questi processi figli e i genitori `WC`: esecutione parallela. L'input proviene dallo stesso processo. Non ci sono comunicazioni fra questi processi figli e i genitori
PQ Distrib	Distribuzione parallel query

Tabella 6.6: Significato dei campi dell'execution plan

`ORDERS` viene scandita serialmente ma i dati vengono partizionati ed inviati a diversi parallel server. Il significato dei campi è riportato in tabella 6.6.

Vediamo di capire cosa cambia al variare dei parametri di parallelismo: la query originale prevede un *full scan* sulla tabella `ORDERS` con l'applicazione del filtro o_custkey>=0 che è un semplice vincolo imposto dai requisiti tpc-h. Viene anche scandita la tabella `CUSTOMER` e le due tabelle vengono joinate scartando le righe che non hanno la stessa custkey. La distribuzione dei compiti è seriale. Le righe joinate sono alla fine 1505000 con un uso di spazio temporaneo di 28MB. I costi sono proporzionali alla gravità dei compiti. Joinare le tabelle rispettando la condizione di where costa molto di più che operare un full scan delle due tabelle.

Se la tabella `CUSTOMER` viene dichiarata parallela, `ORDERS` viene scandita in full scan seriale e inviata in pipe a più processi che ricevono i dati. I processi di ricezione ordinano i dati e si preparano al join. Parallelamente la tabella `CUSTOMER` viene suddivisa in blocchi assegnati a processi paralleli che la scandi-

scono inviandola in pipe. Il join avviene in modo PCWP (input parallelo output parallelo) e il tutto viene spedito al PX query coordinator. La tabella TQ riporta degli identificatori utili per capire l'annidamento delle operazioni.

Se per ultimo dichiariamo anche la tabella `ORDERS` come parallela, otteniamo l'execution plan della tabella 6.7:

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |  1505K|   476M|  1082   (1)| 00:00:13 |        |      |            |
|   1 |  PX COORDINATOR          |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)    | :TQ10002 |  1505K|   476M|  1082   (1)| 00:00:13 |  Q1,02 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN BUFFERED    |          |  1505K|   476M|  1082   (1)| 00:00:13 |  Q1,02 | PCWP |            |
|   4 |     PX JOIN FILTER CREATE| :BF0000  |   150K|    27M|   125   (1)| 00:00:02 |  Q1,02 | PCWP |            |
|   5 |      PX RECEIVE          |          |   150K|    27M|   125   (1)| 00:00:02 |  Q1,02 | PCWP |            |
|   6 |       PX SEND HASH       | :TQ10000 |   150K|    27M|   125   (1)| 00:00:02 |  Q1,00 | P->P | HASH       |
|   7 |        PX BLOCK ITERATOR |          |   150K|    27M|   125   (1)| 00:00:02 |  Q1,00 | PCWC |            |
|   8 |         TABLE ACCESS FULL| CUSTOMER |   150K|    27M|   125   (1)| 00:00:02 |  Q1,00 | PCWP |            |
|   9 |     PX RECEIVE           |          |  1500K|   205M|   955   (1)| 00:00:12 |  Q1,02 | PCWP |            |
|  10 |      PX SEND HASH        | :TQ10001 |  1500K|   205M|   955   (1)| 00:00:12 |  Q1,01 | P->P | HASH       |
|  11 |       PX JOIN FILTER USE | :BF0000  |  1500K|   205M|   955   (1)| 00:00:12 |  Q1,01 | PCWP |            |
|  12 |        PX BLOCK ITERATOR |          |  1500K|   205M|   955   (1)| 00:00:12 |  Q1,01 | PCWC |            |
|* 13 |         TABLE ACCESS FULL| ORDERS   |  1500K|   205M|   955   (1)| 00:00:12 |  Q1,01 | PCWP |            |
-----------------------------------------------------------------------------------------------------------
* access("C_CUSTKEY"="O_CUSTKEY")
* filter("O_CUSTKEY">=0)
```

Tabella 6.7: Execution plan della query con le tabelle `CUSTOMER` e `ORDERS` dichiarate parallele

In quest'ultimo execution plan si può notare che sostanzialmente cambia il modo di accedere alla tabella `ORDERS` che è passato da seriale a parallelo. Altri cambiamenti, rispetto al caso in cui solo `CUSTOMER` è dichiarato parallelo, riguardano le strutture di Oracle necessarie per lo svolgimento in parallelo dei compiti (PX block iterator).

Per la query 1.sql l'execution plan è quello indicato in tabella 6.8.

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     5 |   225 |  9544   (3)| 00:01:55 |        |      |            |
|   1 |  PX COORDINATOR          |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (ORDER)     | :TQ10002 |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,02 | P->S | QC (ORDER) |
|   3 |    SORT ORDER BY         |          |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,02 | PCWP |            |
|   4 |     PX RECEIVE           |          |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,02 | PCWP |            |
|   5 |      PX SEND RANGE       | :TQ10001 |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,01 | P->P | RANGE      |
|   6 |       HASH GROUP BY      |          |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,01 | PCWP |            |
|   7 |        PX RECEIVE        |          |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,01 | PCWP |            |
|   8 |         PX SEND HASH     | :TQ10000 |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,00 | P->P | HASH       |
|   9 |          HASH GROUP BY   |          |     5 |   225 |  9544   (3)| 00:01:55 |  Q1,00 | PCWP |            |
|  10 |           PX BLOCK ITERATOR |       |  5876K|   252M|  9331   (1)| 00:01:52 |  Q1,00 | PCWC |            |
|* 11 |            TABLE ACCESS FULL| LINEITEM |  5876K|   252M|  9331   (1)| 00:01:52 |  Q1,00 | PCWP |            |
-----------------------------------------------------------------------------------------------------------
```

Tabella 6.8: Execution plan della query *1.sql* con tutte le tabelle dichiarate parallele

6.4.3 Il grado di parallelismo

Il grado di parallelismo indica a quanti *parallel execution server* verranno affidati i compiti da eseguirsi su una tabella ed è un parametro delicato che può avere un peso determinante nelle prestazioni. Normalmente Oracle si incarica di determinare questo parametro sulla base delle dichiarazioni delle tabelle, del carico

di lavoro del sistema al momento della richiesta di esecuzione della query e delle impostazioni dei parallel server dell'istanza del database.

Per i nostri scopi non è indicato che il DOP possa variare senza preavviso in quanto cambierebbero le condizioni di valutazione delle prestazioni dei sistemi che andremo a testare. Oracle è studiato per avere prestazioni *ottime* ma per far questo imposta dinamicamente molti parametri che aggiungono troppe variabili allo studio delle prestazioni su sistemi multiprocessore. Per fortuna ci sono metodi impliciti ed espliciti per indicare il DOP per ogni operazione: il parametro PARALLEL_ADAPTIVE_MULTI_USER (tabella 6.3) adatta dinamicamente il DOP per prevenire l'overload del sistema; trattandosi di una valutazione dinamica è bene disabilitare questa opzione per avere la certezza che il DOP non venga modificato in base alle condizioni del sistema. Il parametro PARALLEL_MAX_SERVERS è settato di default a CPU_COUNT x PARALLEL_THREADS_PER_CPU e viene ulteriormente moltiplicato per 5 se PGA_AGGREGATE_TARGET > 0, altrimenti viene moltiplicato per 10. Anche in questo caso occorre tenere presente che la configurazione è dinamica, per quanto un calcolo dinamico sulla base dei processori può abbassare il tempo di setup ed essere quindi utile per testare configurazioni con diversi numeri di processori.

6.4.4 Ottimizzazione

Gli obiettivi che ci prefiggiamo sono di abbattere il più possibile gli accessi al disco e di rendere le operazioni parallele. Oracle dispone di due opzioni: *cache* e *parallel* che rispettivamente indicano al motore di esecuzione di porre le tabelle coinvolte nelle query in una coda di tipo MRU[6] e di parallelizzare le operazioni su di esse.

Opzione *cache*

Come scelta predefinita, quando Oracle effettua un full scan di una tabella, pone i dati in una coda LRU[7]. Questa scelta è giustificata dal fatto che normalmente i dati acquisiti in una scansione totale di una tabella vengono utilizzati solo al momento della lettura. L'opzione *cache* permette di ripristinare una coda MRU. Per le dimensioni assunte per il database questo comporta abbattere gli accessi al disco durante la seconda esecuzione, come evidenziato dal grafico di figura 6.2 i cui dati sono stati ricavati impostando per la tabella *lineitem* oltre che l'opzione *cache*, l'opzione *noparallel* (vedi sezione 6.4.4).

Il comando SQL per abilitare l'opzione *cache* in Oracle è il seguente:

```
SQL> ALTER TABLE LINEITEM CACHE;
```

[6]Most recently used
[7]Least recently used

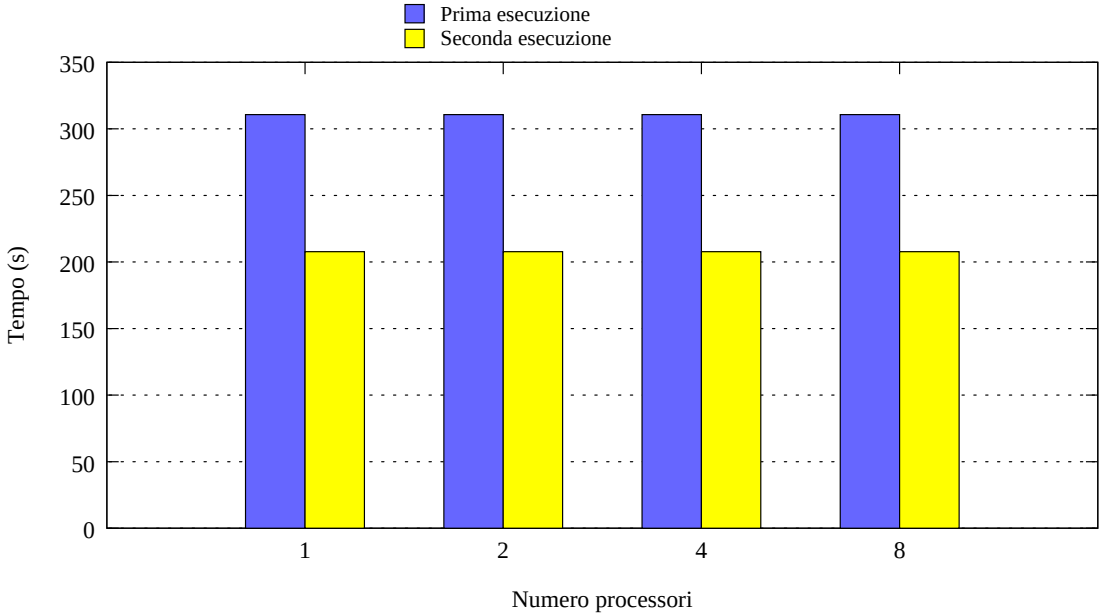

Figura 6.2: Tempi di prima e seconda esecuzione con opzioni cache e noparallel settate

Opzione *parallel*

L'opzione *parallel* abilita l'esecuzione parallela delle operazioni compiute su una certa tabella. L'opzione è a livello di tabella e può essere indicato il grado di parallelismo che altrimenti viene calcolato dimanicamente da Oracle, eventualmente sulla base del carico di lavoro della macchina al momento dell'esecuzione della query. Il comando per specificare l'opzione è il seguente:

```
SQL> ALTER TABLE LINEITEM PARALLEL;
```

Come si evince dalla figura 6.3, i tempi della seconda esecuzione della query 1.sql presentano un valore che scala linearmente al crescere dei processori.

Una volta ottimizzati i valori dei parametri per Oracle il tempo di esecuzione delle query ha l'andamento del grafico in figura 6.4. La relazione lineare fra il numero di processori e il tempo di esecuzione indica che è stata effettuata una parallelizzazione ottimale delle query.

6.4.5 I processi Oracle su Solaris

Solaris permette di impostare su quali processori verranno eseguiti determinati processi (questa cosa viene anche chiamata *affinity*). Mediante il comando *mpstat* è possibile ottenere, oltre alle statistiche di uso, gli *id* dei processori (nell'esempio della tabella 6.9 per una macchina con 4 processori):

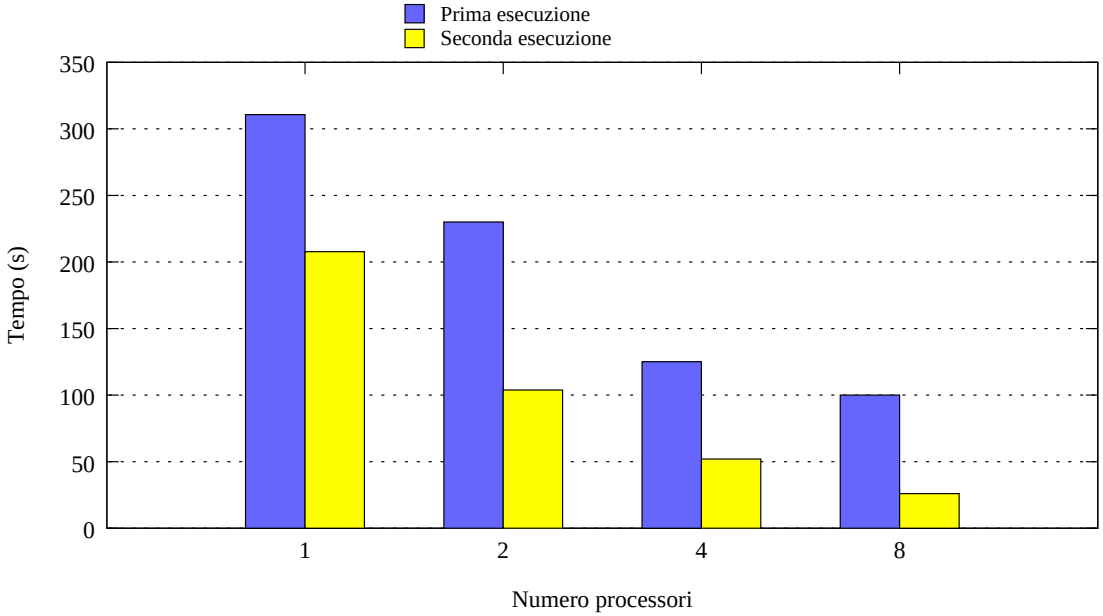

Figura 6.3: Prima e seconda esecuzione con opzioni cache e parallel abilitate

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
0	72	0	321	423	322	75	2	4	6	0	253	1	2	0	97
1	70	0	19	121	110	72	2	4	6	0	256	1	1	0	98
6	71	0	22	105	102	86	2	4	6	0	278	1	1	0	96
7	108	3	23	166	161	100	5	4	7	0	354	2	2	0	96

Tabella 6.9: Esecuzione del comando `mpstat`

Usando il comando *psrset* è possibile creare un set di processori e definire su quale esclusivo set possono girare i vari processi. *Psrset* è illustrato nelle man pages di Solaris alla sezione '1M' che tratta i comandi che vengono usati per la manutenzione del sistema e per scopi amministrativi.

Per allocare i processi creati da Oracle si può procedere in questo modo:

1. avviare l'istanza del database Oracle;
2. identificare i processi Oracle con *ps* (il nome dei processi parallel server è del tipo: `ora_p001`);
3. effettuare il binding dei processi ai set di processori.

Alla luce di quanto visto nel paragrafo 6.4.1 possiamo, ad esempio, scegliere di creare un numero di processi pari al doppio dei processori disponibili. Per limitazioni sul binding dei processi di Solaris, che riserva un processore per operazioni del sistema operativo, conviene creare un numero di processi di Oracle pari al doppio del numero di processori *realmente* disponibili. Le prove sono state effettuate con 7 processori disponibili e quindi 14 processi di Oracle.

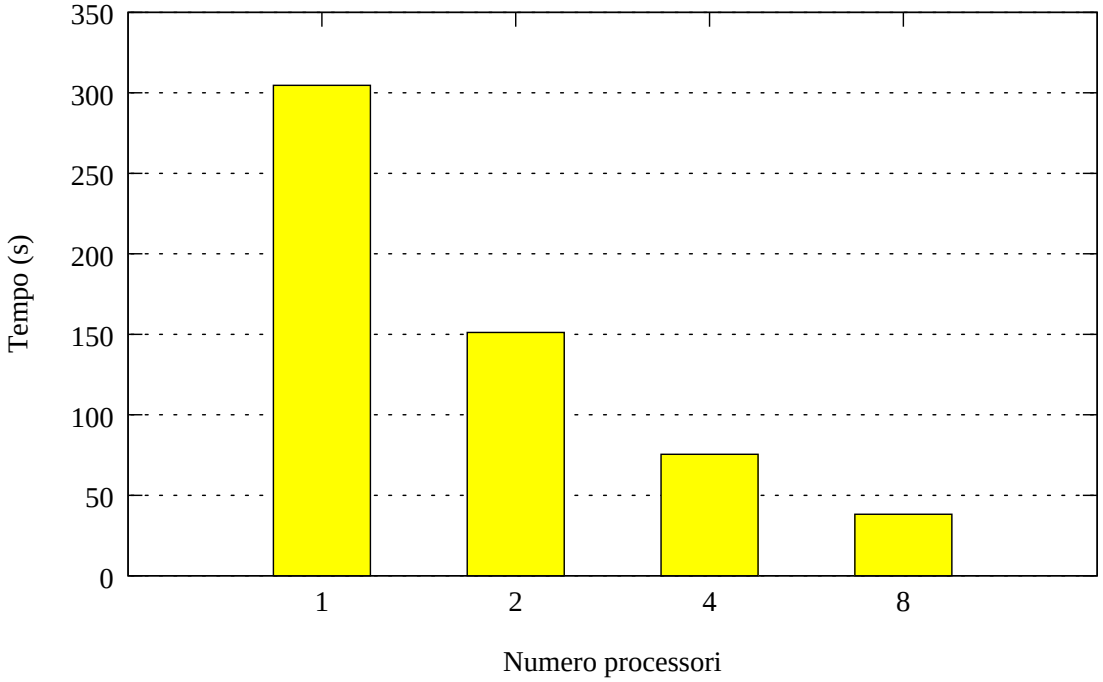

Figura 6.4: Tempo di esecuzione per numero processori

Per assicurarsi che il numero di processi creati da Oracle non vari dinamicamente, possiamo settare allo stesso valore il numero minimo e massimo di parallel server (vedi tabella 6.3).

6.4.6 Binding dei processi

In un sistema multiprocessore i processi possono migrare da un processore ad un altro per scelta dello schedulatore. Questo movimento introduce troppe variabili allo studio del comportamento delle cache per cui abbiamo provveduto a creare un *bind* fra i processi di Oracle, il cui numero è stato fissato nel paragrafo 6.4.5, e i processori, riservandone uno per il sistema operativo (figura 6.5).

6.5 Passaggio alla simulazione microarchitetturale

Data l'estrema lentezza che caratterizza la simulazione microarchitetturale è opportuno commutare il simulatore dalla simulazione funzionale a quella architetturale nel momento più prossimo possibile all'inizio dell'esecuzione della query con la quale si intende effettuare il benchmark. Operando ad alto livello, le possibilità di individurare questo istante sono limitate ma Oracle permette l'esecuzione di procedure esterne durante lo svolgimento di una query.

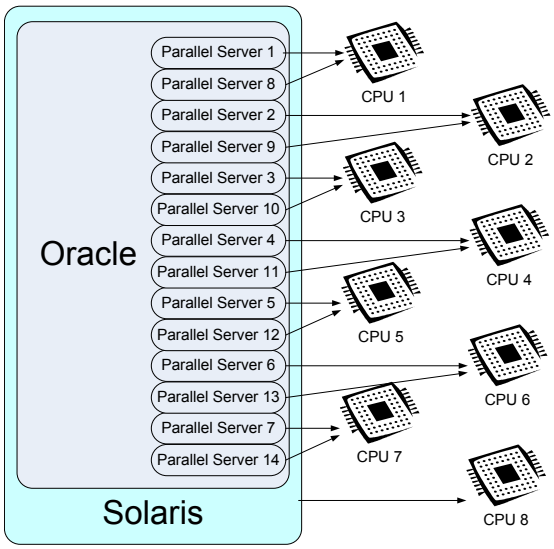

Figura 6.5: Binding dei processi di Oracle

L'idea è quella di creare un programma contenente una chiamata alla *magic instruction* che interromperà la simulazione funzionale. L'handler può venire scritto per contenere i comandi che permettano il passaggio alla simulazione microarchitetturale.

Il programma in C compilato per Solaris è il seguente:

```
1  #include<magic-instruction.h>
2
3  void *global_data;
4  int main()
5  {
6    MAGIC(1);
7    exit(0);
8  }
```

Tale programma differisce da quello presentato in 3.5 nell'ultima istruzione, che è un *explicit exit*. Lo schedulatore di Oracle, preposto all'esecuzione dei programmi esterni, si comporta in modo non definito quando questa chiamata non è specificata, per cui è bene che l'ultima istruzione eseguita prima della fine del programma sia una *exit*. A causa di un bug non risolto alla data attuale, Oracle è in grado di eseguire solo i programmi di proprietà dell'utente *nobody*, per cui dovremo assegnare il programma *magic_ instruction* a questo utente mediante il comando:

```
# chown nobody magic_instruction
```

Oracle permette l'esecuzione di programmi esterni mediante il suo stesso schedulatore. Per far questo occorre assegnare all'utente *oracle* i privilegi per creare job esterni nello schedulatore, cosa che si fa loggandosi come utente *sys*:

```
SQL*Plus: Release 10.2.0.1.0 - Production on Gio Mag 18 11:48:48 2006

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> GRANT CREATE EXTERNAL JOB TO "ORACLE";

Concessione riuscita.
```

L'utente *oracle* può ora programmare lo schedulatore per l'esecuzione del programma esterno:

```
1   BEGIN
2   sys.dbms_scheduler.create_job(
3   job_name => "ORACLE"."MAGIC_CALL",
4   job_type => EXECUTABLE,
5   job_action => /oracle/magic_instruction2,
6   start_date => systimestamp at time zone Europe/Rome,
7   job_class => DEFAULT_JOB_CLASS,
8   comments => chiamata alla magic instruction,
9   auto_drop => FALSE,
10  enabled => FALSE);
11  sys.dbms_scheduler.set_attribute(
12  name => "ORACLE"."MAGIC_CALL",
13  attribute => logging_level,
14  value => DBMS_SCHEDULER.LOGGING_OFF);
15  sys.dbms_scheduler.set_attribute(
16  name => "ORACLE"."MAGIC_CALL",
17  attribute => restartable,
18  value => TRUE);
19  sys.dbms_scheduler.enable( "ORACLE"."MAGIC_CALL" );
20  END;
21  /
```

La query originale può essere modificata per contenere la chiamata al job esterno appena definito:

```
1   exec dbms_scheduler.run_job(VMSTAT_JOB);
2   select
3     l_returnflag,
4     l_linestatus,
5     sum(l_quantity) as sum_qty,
6     sum(l_extendedprice) as sum_base_price,
7     sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
8     sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
9     avg(l_quantity) as avg_qty,
10    avg(l_extendedprice) as avg_price,
11    avg(l_discount) as avg_disc,
12    count(*) as count_order
13  from
14    lineitem
15  where
16    l_shipdate <= date '1998-12-01' - interval '[DELTA]' day (3)
17  group by
18    l_returnflag,
19    l_linestatus
20  order by
21    l_returnflag,
22    l_linestatus;
```

In Simics l'esecuzione del comando shell:

```
# ./sqlplus "oracle/oracle" < /oracle/query/1.sql
```

provoca il caricamento di *sqlplus* e l'esecuzione della query 1.sql. La simulazione viene interrotta all'esecuzione della magic instruction e Simics restituisce il prompt dei comandi:

```
[cpu8] v:0x00000000000010668 p:0x0000000000420c0668 magic (sethi 0x1, %g0)
simics>
```

A questo punto è possibile commutare simics alla simulazione microarchitetturale con Gems eseguendo:

```
simics> run-command-file simics_command_nuca
```

simics_command_nuca contiene i comandi di inizializzazione di Gems che sono i seguenti:

```
istc-disable
dstc-disable
instruction-fetch-mode instruction-fetch-trace
load-module ruby
```

```
load-module opal
ruby0.setparam g_NUM_PROCESSORS 8
ruby0.setparam g_PROCS_PER_CHIP 8
ruby0.setparam g_NUM_MEMORIES 8
ruby0.setparam_str g_NETWORK_TOPOLOGY FILE_SPECIFIED
ruby0.setparam_str g_DYNAMIC_TIMEOUT_ENABLED false
ruby0.setparam_str REMOVE_SINGLE_CYCLE_DCACHE_FAST_PATH true
ruby0.setparam_str g_CACHE_DESIGN NUCACOL
ruby0.setparam_str g_adaptive_routing false
ruby0.setparam NUMBER_OF_VIRTUAL_NETWORKS 7
ruby0.setparam g_endpoint_bandwidth 1000
ruby0.setparam_str g_NUCA_PREDICTOR_CONFIG SNUCA
ruby0.setparam g_NUM_L2_BANKS 32
ruby0.setparam g_MEMORY_SIZE_BYTES 4294967296
ruby0.init
opal0.init
```

Simics risponde nel modo seguente all'esecuzione del file *simics_ command_ nuca*:

```
simics> run-command-file simics_commands_nuca
Turning I-STC off and flushing old data
Turning D-STC off and flushing old data
successful installation of the ruby timing model.
Queue registration cpu8
successful installation of the opal queue.
hfa_init_local done:
Ruby Timing Mode
Warning: optimizations not enabled.
Creating event queue...
Creating event queue done
Creating system...
  Processors: 8
Creating system done
Ruby: ruby-opal link established. removing timing_model.
OpalInterface: installation successful.
Ruby initialization complete
Ruby: ruby-opal link established. removing timing_model.
opalinterface: doing notify callback
Opal: opal-ruby link established.
OpalInterface: installation successful.
simics>
```

Per avviare la simulazione con Gems a questo punto basta inviare i comandi:

```
opal0.sim-start "statistiche"
opal0.sim-step 1000000
```

Viene avviata la simulazione scrivendo nel file *statistiche* i dettagli della configurazione del simulatore ed eseguendo un milione di step che, nel caso di simulazione in ambiente Opal, corrispondono a un milione di istruzioni eseguite dal primo dei processori definiti nella configurazione della macchina target.

6.6 Statistiche

Simics, Ruby ed Opal generano statistiche delle simulazioni che possono essere estratte per la creazione di grafici. Come il microscopio per il biologo o l'oscilloscopio per l'elettronico questi dati rappresentano lo strumento fondamentale per l'analisi delle prestazioni delle cache e includono informazioni sul numero di istruzioni eseguite, sulle *cache hit* e *cache miss*, informazioni sugli accessi, sulla migrazione dei blocchi e sulla coerenza.

Le informazioni collezionabili diventano più specifiche spostandosi da Simics verso Opal. Per avere una stampa delle statistiche di Opal basta usare il comando:

```
simics> opal0.stats
```

Opal stamperà un numero enorme di statistiche (oltre 1700 righe) che non si trascrivono per motivi di spazio. Purtroppo la maggior parte delle voci presenti nelle statistiche non sono documentate, ma i loro nomi sono altamente esplicativi.

Per stampare le statistiche di Ruby possiamo usare il comando (oltre 2400 righe)

```
ruby0.dump-stats
```

oppure, accontentandosi di un sottoinsieme molto ristretto di statistiche, possiamo usare i più comodi comandi

```
ruby0.periodic-stats-file
ruby0.periodic-stats-interval
```

che consentono di ridirigere una stampa periodica di statistiche su un file per la successiva analisi o estrazione mediante strumenti standard Unix, quali *grep* o *awk*. Ad esempio per estrarre le statistiche da Ruby circa le istruzioni eseguite ogni 10000 step, basta impostare

```
simics> ruby0.periodic-stats-interval 10000
simics> ruby0.periodic-stats-file "statistiche"
simics> c
```

al termine della simulazione (oppure, volendo, anche al momento in cui si crede che le statistiche bastino), si può filtrare dal file di statistiche il parametro che ci interessa, per esempio *instruction_ executed*[8], con:

```
$ grep instruction_executed statistiche | awk '{ print $2,",",$4,
  ",",$5,",",$6,",",$7,",",$8,",",$9,",",$10,",",$11}'
```

il risultato è la stampa a video di un testo CSV[9] compatibile che può essere importato nei più comuni fogli di calcolo.

[8]numero di istruzioni eseguite
[9]Comma Separated Values

CAPITOLO 7

Simulazioni

Una volta ottimizzato il database e fissata l'affinità dei processi di Oracle ai processori disponibili nel modo più equo possibile, le simulazioni permettono di verificare un bilanciamento del carico di lavoro sui processori: Sulle ascisse del

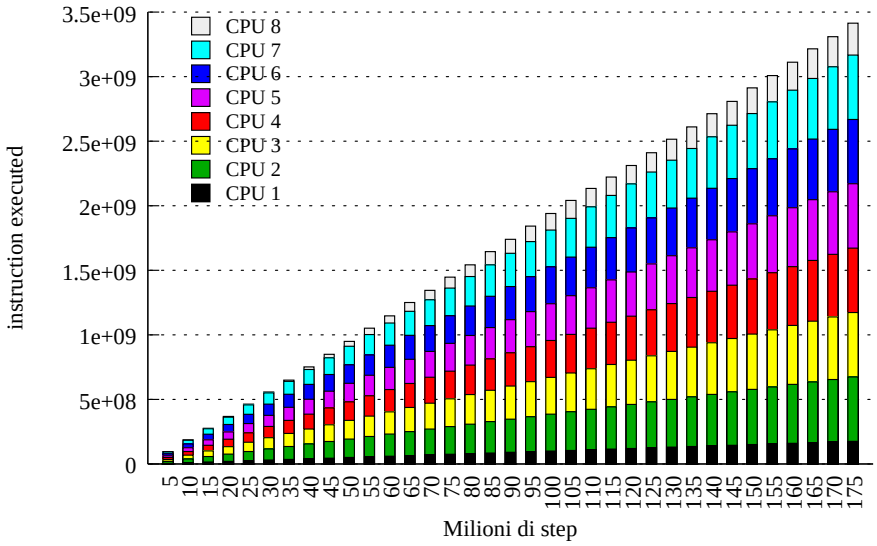

Figura 7.1: Numero di istruzioni eseguite per processore durante lo svolgimento della query *1.sql*

grafico 7.1 sono riportati il numero di step di Opal, equivalenti al numero di milioni di istruzioni eseguite dal primo processore del set. I segmenti dei grafici a barre rappresentano il numero di istruzioni eseguite da ogni singolo processore. La

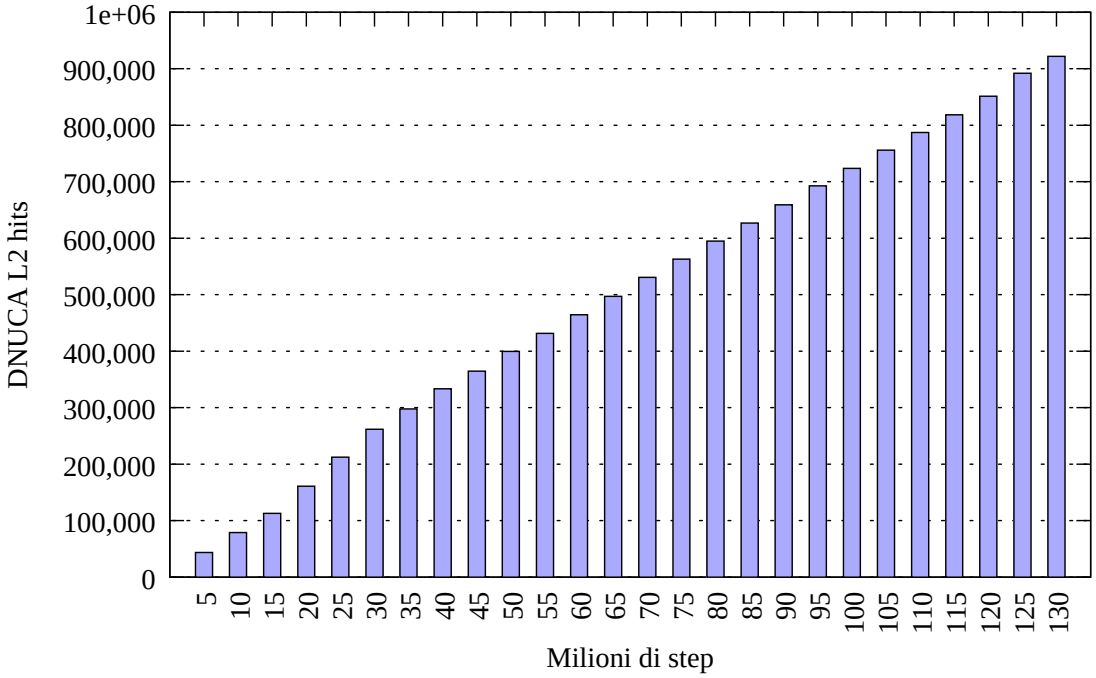

Figura 7.2: Andamento delle hit L2 per un singolo processore durante l'esecuzione della query *1.sql*

linearità della distribuzione conferma che il bilanciamento è stato correttamente eseguito.

È possibile realizzare grafici di tutti i parametri riportati nelle statistiche di Simics, Ruby o Opal. In figura 7.2 è riportato il grafico che mostra il numero di hit sulla cache L2 al variare del numero di step. Il grafico, come tutti i successivi, si riferisce all'esecuzione della query 1 che abbiamo trattato nel paragrafo 6.3. Il cambio di pendenza che si nota verso il 30 milionesimo step corrisponde al termine del caricamento del *working set* del processo.

Nella distribuzione delle miss per processore del grafico in figura 7.3, vediamo che queste sono maggiori nel primo e nell'ultimo processore. Questo a causa del fatto che, nella simulazione, a questi due processori sono stati assegnati i compiti del sistema operativo. In figura 7.4 vediamo l'effetto che un cambio di contesto provoca nel grafico del rapporto fra *istruzioni cache L2* e *miss*. Il cambio di contesto si ha in vicinanza del 110 milionesimo step. Nella configurazione usata per questa prova ai processi Oracle era permesso di migrare.

Alla stessa simulazione si riferiscono anche i grafici delle figure 7.5 e 7.6, nelle quali vediamo l'effetto che ha il cambio di contesto sul numero delle miss totali (figura 7.5) e conseguentemente sul numero di rimpiazzamenti (figura 7.6).

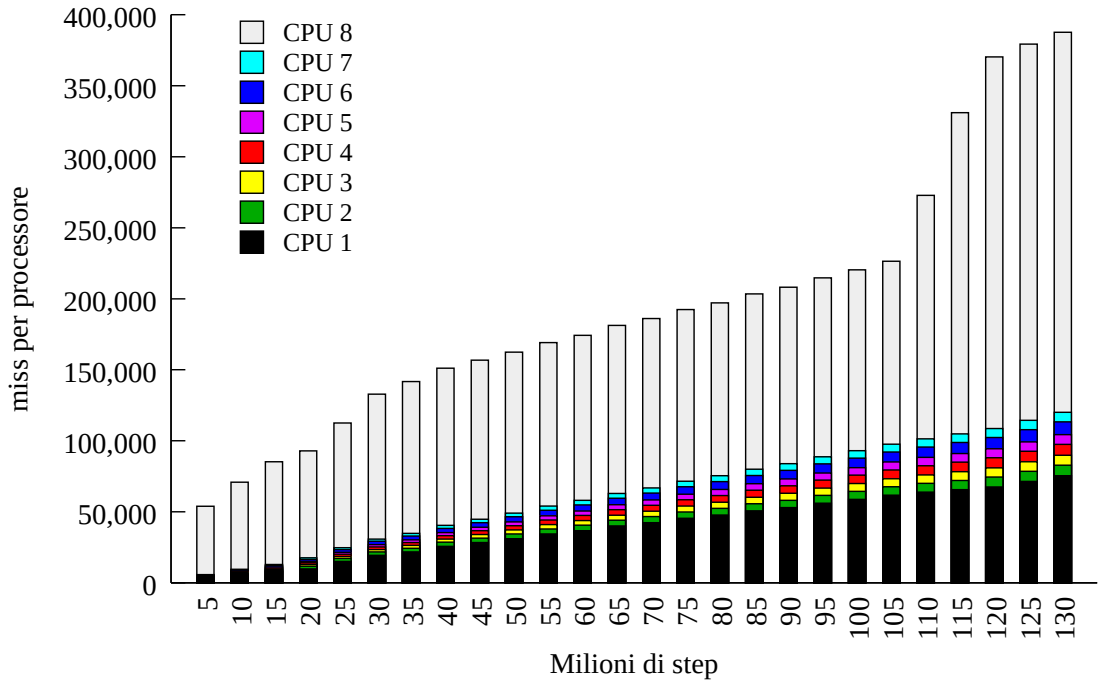

Figura 7.3: Andamento delle miss della cache L2 per un singolo processore durante l'esecuzione della query *1.sql*

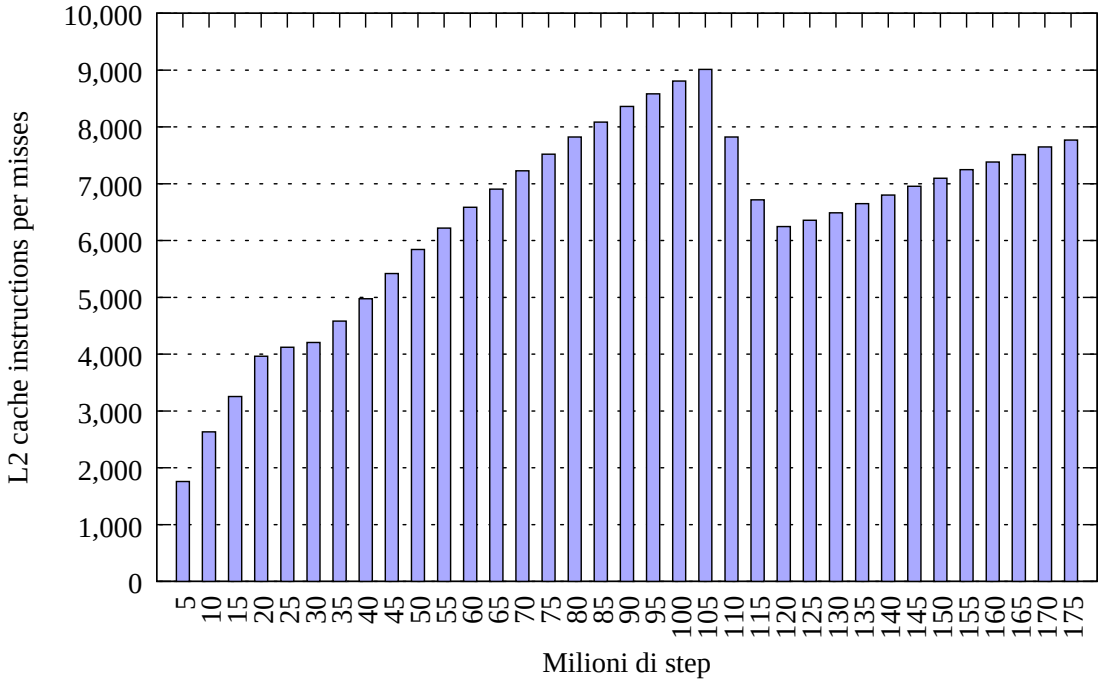

Figura 7.4: Andamento del rapporto istruzioni/miss della cache L2 durante l'esecuzione della query *1.sql*

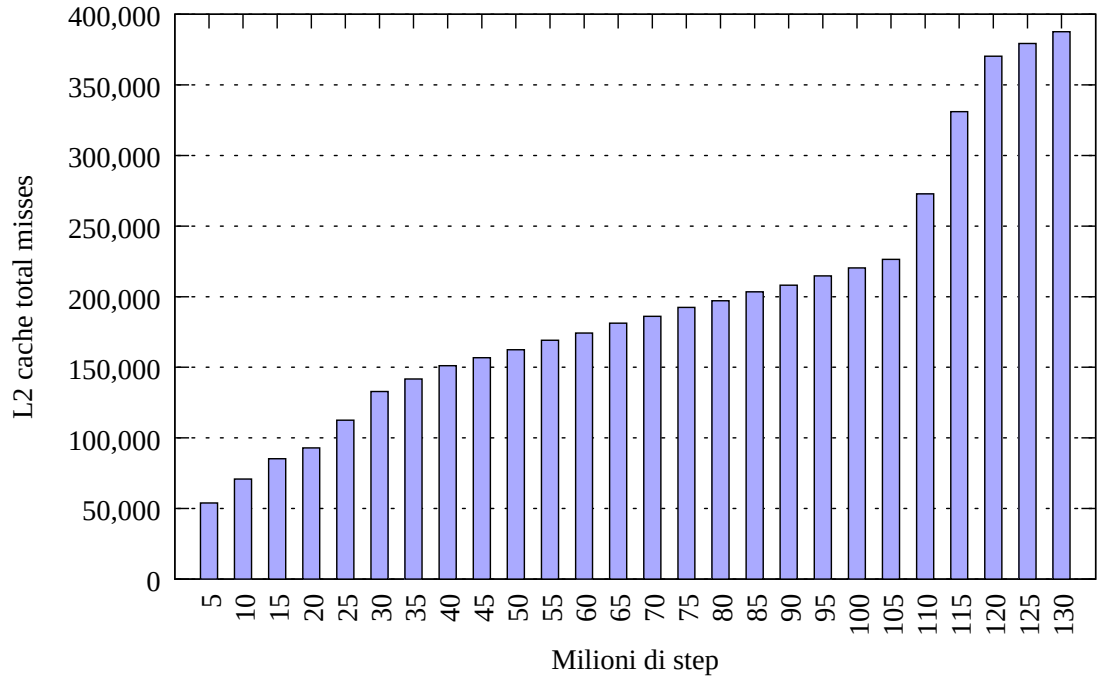

Figura 7.5: Miss totali nella cache L2 durante l'esecuzione della query *1.sql*

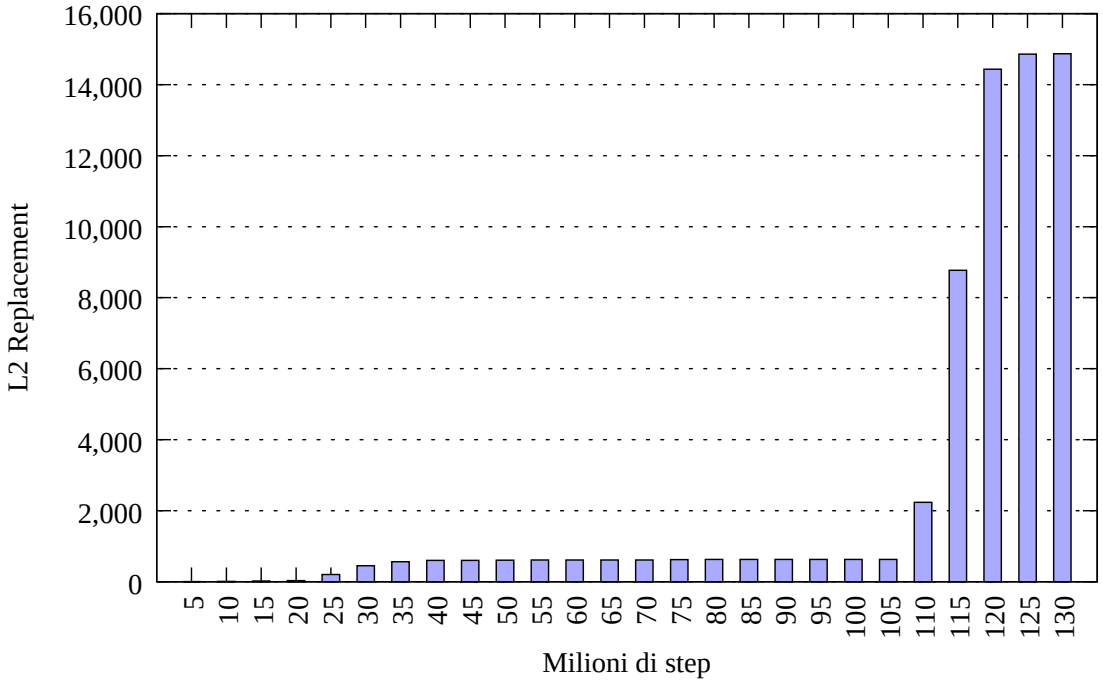

Figura 7.6: Andamento del numero totale dei rimpiazzamenti nella cache L2 durante l'esecuzione della query *1.sql*

CAPITOLO 8

Conclusioni

L'ambiente creato si presenta sotto forma di immagini di dischi e checkpoint. I checkpoint sono immediatamente utilizzabili da Simics e descrivono una macchina target pronta ad eseguire le query tpc-h. Le immagini dei dischi permettono il boot di Solaris e l'avvio dell'istanza di Oracle con qualsiasi simulatore in grado di leggere block device. È possibile valutare le prestazioni di una grande varietà di configurazioni: le variabili modificabili sono il numero di processori, la dimensione e la topologia delle cache, il timing delle memorie, la distribuzione dei processori e delle memorie sulle schede nel backplane, la quantità di ram disponibile e di cache in numero virtualmente arbitrario di livelli. È inoltre possibile descrivere mediante il linguaggio SLICC i protocolli di coerenza di nuovi tipi di cache.

I benchmark TPC-H rappresentano un valido strumento per la valutazione delle prestazioni globali di un sistema e l'ambiente preparato permette di eseguirli senza ulteriori settaggi. I risultati, confrontabili con quelli già presentati dai maggiori vendor hardware, permettono di avere un quadro immediato delle prestazioni senza che sia necessario ripetere i test su altre macchine di prova.

Attualmente la velocità di simulazione rappresenta ancora un problema, soprattutto dopo il passaggio alla simulazione microarchitetturale. I tempi di simulazione sono ancora troppo elevati e costringono a valutare porzioni di esecuzione, per quanto queste possano, in alcuni casi, essere già molto significative. C'è da dire che Simics alla data della pubblicazione di questa tesi non è multiprogrammato e non può quindi beneficiare delle macchine host multiprocessore. Inoltre i moduli Ruby ed Opal sono stati fatti non ricorrendo all'interfaccia MAI[1] che è il modo più efficiente per interfacciarsi con Simics.

È realistico pensare, visto l'interesse crescente per le cache NUCA, che presto

[1]Micro-Architectural Interface

Virtutech implementerà propri moduli equivalenti a GEMS che ridurranno molto lo *slowdown* della simulazione micro-architetturale. Fino ad allora conviene concentrarsi, come viene attualmente fatto in molte pubblicazioni, nel determinare intervalli di esecuzione che possano essere rappresentativi.

APPENDICE A

Sistemi utilizzati

A.1 Sistemi usati

Per questo lavoro di tesi sono state utilizzate le macchine riportate in tabella A.1 Data la maggior potenza di calcolo, vega è stato il server usato maggiormente per

Nome macchina	**Configurazione**
vega	Server dual Xeon Debian GNU/Linux 4GB ram 240GB disco fisso
sun	Workstation Sparc Ultra 10 Solaris 10 1GB ram 20GB disco fisso
chomp	Workstation P4 dual-core Windows XP 2GB ram 1TB disco fisso

Tabella A.1: Sistemi Utilizzati

le simulazioni con Simics e per eseguire l'installazione di Solaris sulla macchina target. È stato inoltre usato per il parsing delle statistiche e per la generazione dei grafici.

Sun è una macchina con processore Sparc ed è stata usata, data la maggior

velocità di risposta rispetto ad una macchina simulata, per mettere a punto le procedure di configurazione di Solaris ed Oracle.

Chomp è un'unità di supporto ed è stata usata per eseguire simulazioni minori oltre che per redigere questa tesi.

A.2 Note di installazione

Simulare sistemi multiprocessore richiede molto tempo ed è opportuno ingegnarsi per trovare il modo di risparmiarlo. Una giusta strategia che si avvalga di sistemi dedicati può consentire un enorme risparmio di tempo.

A.2.1 Simics

Al momento della pubblicazione di questa tesi Simics è aggiornato alla versione 3. Purtroppo GEMS può funzionare soltanto con la versione 2 di Simics. La versione 3 permette di alterare con grande facilità le configurazioni delle macchine target. È stato quindi usato Simics 3 per la configurazione delle macchine sulle quali effettuare l'installazione del software di base (Solaris ed Oracle). Le immagini dei dischi sono state successivamente passate a GEMS/Simics 2 per la simulazione microarchitetturale.

A.2.2 Solaris

Per l'installazione di Solaris è stato sfruttato il fatto che la riconfigurazione del sistema operativo può essere fatta dinamicamente al boot; esiste infatti un'apposita opzione da passare ad OpenBoot, l'equivalente Sun del BIOS nei sistemi x86. Questo ha permesso di installare Solaris su una macchina con un solo processore e 256 MB di ram e successivamente di cambiare la configurazione della macchina. Passando ad OpenBoot il comando:

```
boot-command "boot disk1 -rv"
```

si comanda la riconfigurazione di Solaris con il nuovo hardware installato.

Il risparmio di tempo che si è ottenuto riconfigurando la macchina target anziché procedere ad una nuova installazione del sistema operativo è di 1-2 giorni per ogni diversa configurazione di cpu che abbiamo considerato (1, 2, 4 o 8 processori).

A.2.3 Oracle

Oltre che sulla macchina target, Oracle è stato installato anche in una macchina Windows dual-core. Sulla macchina Windows è stato eseguito l'oneroso compito di importare le tabelle generate da `qgen`. Successivamente la base di dati è stata

esportata con l'utility Oracle '`exp`' ed importata sulla macchina target con l'utility '`imp`' (figura A.1).

Spostare questo processo su una macchina reale ha permesso un risparmio di tempo di circa 7 ore.

Prima di sottoporre al simulatore le query, tutte le procedure sono state testate su una macchina reale Sun Ultra 10 in modo da evitare errori che in ambienti simulati avrebbero portato ad enormi perdite di tempo.

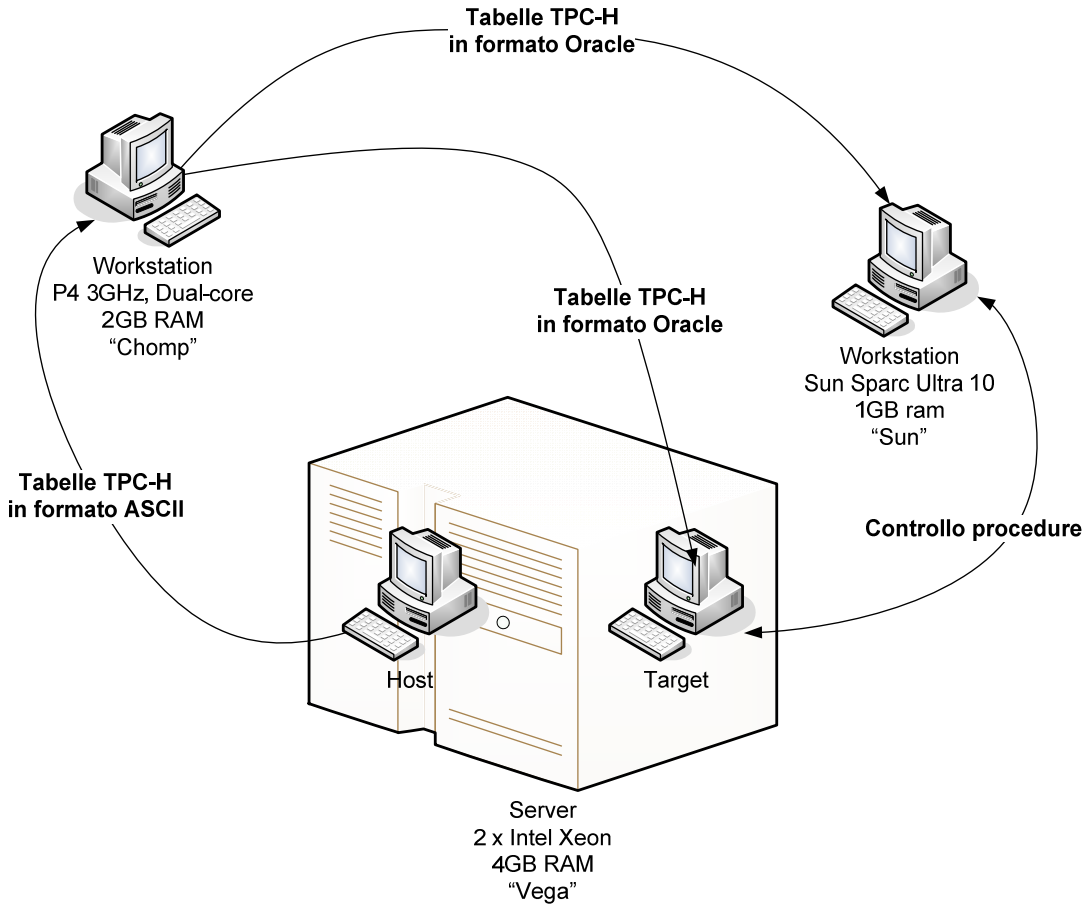

Figura A.1: Utilizzo delle macchine disponibili

APPENDICE B

Sun Enterprise 6500 Server

Il Sun Enterprise 6500 (figura B.1) è un server basato su un *backplane*[1] operante a 2.68GB/s che mette a disposizione 16 slot per l'installazione di schede processori ed interfacce. Ogni scheda processore ha a disposizione 2 slot per l'installazione di cpu e può alloggiare fino a 4 GB di ram su 16 DIMM. Le cpu installabili sono Sparc v9 superscalari con 32 KB di cache L1 e da 4 a 8 MB di cache L2. La configurazione minima prevede una scheda cpu ed una scheda di I/O; lo chassis del server misura 173 x 77 x 99 cm e pesa 450Kg. In configurazione completa (30 cpu) il server consuma circa 5 kW.

Durante la configurazione di Simics si è fatta la scelta di distribuire uniformemente sulle schede cpu la memoria ram, come riportato in tabella B.1. Occorre

Numero processori	**Numero schede cpu**	**Distribuzione ram**
1	1	1 x 4 GB
2	1	1 x 4 GB
4	2	2 x 2 GB
8	4	4 x 1 GB

Tabella B.1: Distribuzione della ram

tenere presente che, quando si esegue Gems, il modello di distribuzione della ram nell'implementazione cambia e diventa indipendente dalla configurazione impostata.

Simics emula le schede dell'Enterprise 6500 mettendo a disposizione una serie di oggetti con i relativi connettori software. Di seguito si riporta la lista dei com-

[1]sistema di interconnessione

Figura B.1: Sun Enterprise 6500 Server

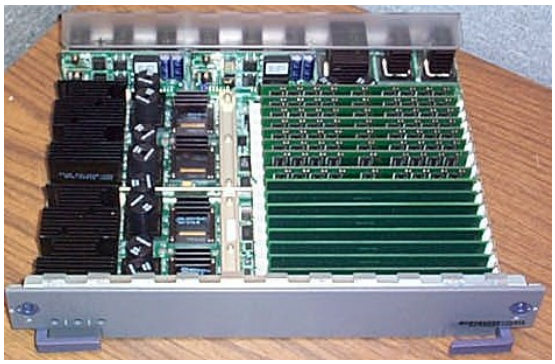

Figura B.2: La scheda CPU di un server Sun Enterprise

Figura B.3: La tastiera di un server Sun Enterprise

ponenti istanziati da Simics per la corretta esecuzione dell'ambiente che abbiamo preparato:

```
simics> list-components
scsi_cdrom_cmp0              - std-scsi-cdrom         (top: system_cmp0)
-------------------------------------------------------------------------
   scsi-bus        scsi-bus             up       scsi_bus_cmp0:scsi-bus

text_console_cmp0            - std-text-console       (top: system_cmp0)
-------------------------------------------------------------------------
   serial          serial               up       system_cmp0:ttya

service_node_cmp0            - std-service-node       (top: system_cmp0)
-------------------------------------------------------------------------
   link0           ethernet-link        up       ethernet_link_cmp0:device

ethernet_link_cmp0           - std-ethernet-link      (top: system_cmp0)
-------------------------------------------------------------------------
   device          ethernet-link        any      board_cmp1:ethernet
                                                 service_node_cmp0:link0

board_cmp1                   - sunfire-sbus-board     (top: system_cmp0)
-------------------------------------------------------------------------
   backplane       sunfire-backplane    up       system_cmp0:slot1
   ethernet        ethernet-link        down     ethernet_link_cmp0:device
   scsi-bus        scsi-bus             down     scsi_bus_cmp0:scsi-bus
   slot0           sun-sbus             down     <empty>
   slot1           sun-sbus             down     <empty>
   slot2           sun-sbus             down     <empty>

board_cmp0                   - sunfire-cpu-board      (top: system_cmp0)
-------------------------------------------------------------------------
   backplane       sunfire-backplane    up       system_cmp0:slot0
   cache-cpu0      timing-model         down     <empty>
   cache-cpu1      timing-model         down     <empty>
   central-cpu     sunfire-central-cpu  up       system_cmp0:central-cpu

scsi_bus_cmp0                - std-scsi-bus           (top: system_cmp0)
-------------------------------------------------------------------------
   scsi-bus        scsi-bus             any      board_cmp1:scsi-bus
                                                 scsi_disk_cmp0:scsi-bus
                                                 scsi_cdrom_cmp0:scsi-bus

scsi_disk_cmp0               - std-scsi-disk          (top: system_cmp0)
-------------------------------------------------------------------------
   scsi-bus        scsi-bus             up       scsi_bus_cmp0:scsi-bus

system_cmp0                  - sunfire-6500-backplane (top: system_cmp0)
-------------------------------------------------------------------------
   central-cpu     sunfire-central-cpu  down     board_cmp0:central-cpu
   keyboard        serial               down     <empty>
```

```
mouse           serial              down    <empty>
remote_console  serial              down    <empty>
slot0           sunfire-backplane   down    board_cmp0:backplane
slot1           sunfire-backplane   down    board_cmp1:backplane
slot10          sunfire-backplane   down    <empty>
slot11          sunfire-backplane   down    <empty>
slot12          sunfire-backplane   down    <empty>
slot13          sunfire-backplane   down    <empty>
slot14          sunfire-backplane   down    <empty>
slot15          sunfire-backplane   down    <empty>
slot2           sunfire-backplane   down    <empty>
slot3           sunfire-backplane   down    <empty>
slot4           sunfire-backplane   down    <empty>
slot5           sunfire-backplane   down    <empty>
slot6           sunfire-backplane   down    <empty>
slot7           sunfire-backplane   down    <empty>
slot8           sunfire-backplane   down    <empty>
slot9           sunfire-backplane   down    <empty>
ttya            serial              down    text_console_cmp0:serial
ttyb            serial              down    <empty>
```

Numero di processori	da 1 a 30
Architettura	Superscalare SPARC Version 9, Ultra-SPARC
Cache per processore	L1: 16-KB instruction, and 16-KB data on chip L2: 4 or 8 MB esterna
Interfacce CPU	da 1 a 30, 128-bit Ultra Port Architecture (UPA) slot
Sistema di interconnessione	Gigaplane, 2.68 GB/sec (84 MHz)

Tabella B.2: Caratteristiche Enterprise server 6500

APPENDICE C

Glossario

ACID Nell'ambito dei database, ACID deriva dall'acronimo inglese Atomicity, Consistency, Isolation, e Durability (Atomicità, Consistenza, Isolamento e Durabilità)

BIOS Basic Input/Output System - Basic Integrated Operating System

CSV Comma Separated Values

DBMS Database Management System

DOP Degree of Parallelism

GEMS General Execution-driven Multiprocessor Simulator

HTML HyperText Markup Language

ISO International Organization for Standardization

LWP lightweight processes

LRU Least recently used

MAI Micro-Architectural Interface

MRU Most recently used

NUCA Non-Uniform Cache Architecture

OLTP On Line Transaction Processing

PQ Parallel Query

QC Query Coordinator

RDBMS Relational Database Management System

RAM Random Access Memory

ROM Read-Only Memory

SCSI Small Computer System Interface

SLICC Specification Language including Cache Coherence

SGA System Global Area

SQL Sequel Language

TCP/IP Transmission Control Protocol / Internet Protocol

TPC Transaction Processing Performance Council

ZFS Zettabyte File System

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Python programming language. `http://www.python.org`.

[2] Bradford M. Beckmann and David A. Wood. Managing Wire Delay in Large Chip-Multiprocessor Caches. *IEEE Computer Society*, 2004.

[3] Paolo Canali. Sun Solaris 10 - Un ambiente operativo Unix completo e ottimizzato per le Cpu a 64 bit. *Pc-Magazine*, 2005.

[4] Transaction Processing Performance Council. `http://www.tpc.org`.

[5] Transaction Processing Performance Council. Tpc-h. `http://www.tpc.org/tpch/default.asp`.

[6] Alessandro Franceschi. I demoni di solaris e gli script di avvio. `http://openskills.info/infobox.php?ID=164`.

[7] George Koch Kevin Loney. *La guida completa Oracle9i*. McGraw-Hill, 2003.

[8] Mike Marty. Reseased protocols - multifacet gems documentation. `http://www.cs.wisc.edu/gems/doc/wiki/moin.cgi/ReleasedProtocols`.

[9] Kevin Lienenbrugger Michael Clark, Ajaya Durg. Characterization of TPC-H Queries on AMD Athlon™ Microprocessors. *IEEE Computer Society*, 2001.

[10] Sun Microsystems. Sun microsystems inc. home page. `http://www.sun.com`.

[11] Bradford M. Beckmann Michael R. Marty Min Xu Alaa R. Alameldeen Kevin E. Moore Mark D. Hill Milo M. K. Martin1, Daniel J. Sorin and David A. Wood. Multifacets general execution-driven multiprocessor simulator (gems) toolset. *Computer Architecture News (CAN)*, September 2005.

[12] Kevin M. Obenland. Posix in Real-Time. `http://www.xtrj.org/collection/posix_rtos.htm`.

[13] The Multifacet GEMS Development Team. Wisconsin multifacet gems simulator. `http://www.cs.wisc.edu/gems`.

[14] Virtutech. Simics for embedded software development, debugging & testing. `http://www.virtutech.com`.

[15] Virtutech. *Simics/SunFire Target Guide*, 12 2005.

Indice analitico