Red Hat Enterprise Linux 6

Performance Tuning Guide

Ottimizzazione produttività del sottosistema con Red Hat Enterprise Linux 6

Edizione 4.0

Last Updated: 2017-10-12

Red Hat Enterprise Linux 6 Performance Tuning Guide

Ottimizzazione produttività del sottosistema con Red Hat Enterprise Linux 6
Edizione 4.0

Red Hat Esperto del settore

A cura di

Don Domingo

Laura Bailey

Nota Legale

Sommario

La Guida alla ottimizzazione delle prestazioni descrive il metodo attraverso il quale è possibile ottimizzare le prestazioni di un sistema che esegue Red Hat Enterprise Linux 6. Essa documenta altresì i processi di aggiornamento relativi alle prestazioni con Red Hat Enterprise Linux 6. Anche se la guida contiene procedure collaudate, Red Hat consiglia di testare correttamente tutte le configurazioni desiderate in un ambiente di prova prima di implementarle in un ambiente di produzione. Eseguire altresì un backup di tutti i dati e le configurazioni esistenti prima del processo di ottimizzazione.

Indice

CAPITOLO 1. PANORAMICA

La *Performance Tuning Guide* è un documento completo sulla configurazione e ottimizzazione di Red Hat Enterprise Linux. Anche se questa release contiene informazioni utili per le prestazioni di Red Hat Enterprise Linux 5, le istruzioni fornite sono specifiche a Red Hat Enterprise Linux 6.

La guida viene suddivisa in capitoli i quali riportano sottosistemi specifici di Red Hat Enterprise Linux. La *Performance Tuning Guide* si sofferma sui tre temi principali per il sottosistema:

Funzioni

Ogni capitolo descrive funzioni uniche relative alle prestazioni (o implementate diversamente) di Red Hat Enterprise Linux 6. Questi capitoli affrontano altresì gli aggiornamenti di Red Hat Enterprise Linux 6 in grado di migliorare sensibilmente le prestazioni di sottosistemi specifici di Red Hat Enterprise Linux 5.

Analisi

Questa guida elenca anche gli indicatori di prestazione per ogni sottosistema specifico. Valori tipici per questi indicatori sono descritti nel contesto di servizi specifici, aiutando così l'utente a capire il significato reale dei sistemi di produzione.

La *Performance Tuning Guide* mostra altresì metodi diversi per il ripristino dei dati sulle prestazioni (ad esempio il profiling) per un sottosistema. Da notare che alcuni degli strumenti usati per il profiling sono documentati in modo più dettagliato in altre sezioni.

Configurazione

Le informazioni più importanti presenti in questo libro sono quelle relative alla regolazione delle prestazioni di un sottosistema specifico in Red Hat Enterprise Linux 6. La *Performance Tuning Guide* spiega come ottimizzare un sottosistema di Red Hat Enterprise Linux 6 per servizi specifici.

Ricordate che la modifica delle prestazioni di un sottosistema specifico potrebbe impattare negativamente sulle prestazioni di un altro sottosistema. La configurazione predefinita di Red Hat Enterprise Linux 6 è ideale per la *maggior parte* dei servizi in esecuzione con carichi di lavoro *moderati*.

Le procedure presenti nella *Performance Tuning Guide* sono state testate in modo dettagliato dagli ingegneri di Red Hat. Tuttavia Red Hat consiglia all'utente di provare correttamente tutte le configurazioni desiderate in un ambiente di prova sicuro prima di applicarle nei server di produzione. Prima di iniziare un processo di ottimizzazione del sistema, eseguire il backup di tutte le informazioni sulla configurazione e dei dati.

1.1. A CHI È RIVOLTO

Questo libro è idoneo per due tipi di utenti

Esperti Business/Sistema

Questa guida descrive in modo molto dettagliato le funzioni delle prestazioni di Red Hat Enterprise Linux 6, e fornisce informazioni sufficienti per sottosistemi in presenza di carichi di lavoro specifici (sia per impostazioni predefinite che per quelle ottimizzate). Il livello di descrizione usato per le funzioni delle prestazioni con Red Hat Enterprise Linux 6, aiuta gli utenti e gli esperti informatici alla comprensione sull'idoneità di questa piattaforma a fornire servizi che utilizzano un numero elevato di risorse ad un livello accettabile.

Quando possibile la *Performance Tuning Guide* fornisce i collegamenti per una documentazione più dettagliata su ogni funzione. Queste informazioni aiutano i lettori a capire le funzioni in modo da

poter sviluppare una strategia idonea ad una implementazione e ottimizzazione di Red Hat Enterprise Linux 6. Ciò permetterà agli utenti di sviluppare *ed* esaminare eventuali infrastrutture.

Questo livello di informazioni è rivolto a utenti con una conoscenza dettagliata dei sottosistemi Linux e delle reti di livello enterprise.

Amministratori di sistema

Le procedure presenti in questo libro sono idonee per amministratori di sistema con un livello tecnico RHCE [1] (o equivalente, e cioè, con 3-5 anni di esperienza nell'implementazione e gestione dei sistemi Linux). La *Performance Tuning Guide* cerca di riportare un numero consistente di informazioni sugli effetti riscontrabili con ogni tipo di configurazione; ciò significa descrivere ogni tipo possibile di compromesso.

La chiave sulla comprensione corretta di un processo di ottimizzazione delle prestazioni non è su come analizzare e regolare un sottisistema, ma conoscere come bilanciare e ottimizzare un sistema Red Hat Enterprise Linux 6 *per uno scopo ben preciso*. Ciò significa *anche* essere a conoscenza su quali compromessi e restrizioni siano accettabili, quando si implementa un tipo di configurazione da implementare per aumentare le prestazioni di un sistema specifico.

1.2. SCALABILITÀ ORIZZONTALE

Gli sforzi di Red Hat per il miglioramento delle prestazioni di Red Hat Enterprise Linux 6 si concentrano sulla *scalabilità*. Le funzioni in grado di migliorare le prestazioni vengono valutate principalmente in base al loro impatto sulla piattaforma in presenza di diversi carichi di lavoro — cioè, dal solo web server al mainframe del server farm.

Concentrandosi sulla scalabilità, Red Hat Enterprise Linux è in grado di mantenere una certa versatilità per diversi tipi di carichi di lavoro e scopi. Allo stesso tempo ciò significa che, con l'aumento del carico di lavoro, la riconfigurazione dell'ambiente del server è un processo meno proibitivo (in termini di costo e ore di lavoro) e più intuitivo.

Red Hat apporta miglioramenti al Red Hat Enterprise Linux sia sulla *scalabilità orizzontale* che per la *scalabilità verticale*; tuttavia la scalabilità orizzontale è quella presa più in considerazione. Il concetto dietro alla scalabilità orizzontale è quello di usare *computer standard* multipli per distribuire carichi di lavoro molto grandi e migliorare così le prestazioni e l'affidabilità.

In un server farm tipico i computer standard sono server rack-montati 1U e server blade. Ogni computer standard può avere una dimensione minima simile ad un sistema con due socket, anche se alcuni server farm utilizzano sistemi molto grandi con un numero maggiore di socket. Alcune reti di livello enterprise presentano un mix di sistemi grandi e piccoli; in questi casi, i sistemi grandi rappresentano server ad elevate prestazioni (per esempio server del database) mentre i sistemi più piccoli rappresentano server per applicazioni specifiche (per esempio server di posta o web).

Questo tipo di scalabilità semplifica la necessità di espandere l'infrastruttura informatica: una azienda di dimensioni medie con un carico di lavoro appropriato avrà bisogno di due server pizza box per soddisfare i propri requisiti. Con l'aumentare del numero di impiegati e del volume di lavoro ecc, i requisiti informatici aumentano sia in volume che in complessità. Una scalabilità orizzontale permette di aggiungere semplicemente macchine supplementari con una configurazione (nella maggior parte dei casi) simile a quella precedentemente implementata.

Riassumendo, la scalabilità orizzontale aggiunge un livello di astrazione in grado di semplificare l'amministrazione dell'hardware del sistema. Sviluppando una piattaforma Red Hat Enterprise Linux in grado di avere una scalabilità orizzontale, sarà possibile avere un miglioramento delle capacità e delle prestazioni dei servizi IT, aggiungendo semplicemente macchine facilmente configurabili.

1.2.1. Elaborazione parallela

Gli utenti possono trarre beneficio dalla scalabilità orizzontale di Red Hat Enterprise Linux non solo perchè essa semplifica la gestione hardware del sistema; ma anche perchè rappresenta una filosofia di sviluppo idonea dato il trend corrente di sviluppo hardware.

Considerate che le applicazioni enterprise complesse presentano migliaia di compiti da eseguire simultaneamente con diversi metodi di coordinamento. Nel passato i computer avevano un processore single-core per affrontare tutti questi compiti, virtualmente tutti i processori al giorno d'oggi hanno core multipli. Quindi i computer moderni utilizzano core multipli in un singolo socket, trasformando i desktop con socket singoli o portatili in sistemi con processori multipli.

Dal 2010 i processori AMD e Intel dispongono da due a sedici core prevalentemente in server blade e pizza box, i quali possono contenere al giorno d'oggi fino a 40 core. Questi sistemi ad elevate prestazioni e con un costo contenuto, rendono disponibili sistemi con una vasta gamma di caratteristiche e capacità nel mainstream.

Per avere maggiori prestazioni ed utilizzare il sistema al meglio delle capacità, ogni core deve essere occupato. Ciò significa che sarà necessario avere 32 compiti separati in esecuzione per trarre vantaggio da un server blade a 32-core. Se una infrastruttura blade presenta dieci blade a 32-core, allora la struttura stessa è in grado di processare un minimo di 320 compiti simultaneamente. Se questi compiti fanno parte di un evento singolo, essi devono essere coordinati.

Red Hat Enterprise Linux è stato sviluppato per adattarsi bene ai diversi trend di sviluppo hardware, e rendere sempre disponibile le migliori prestazioni possibili. Sezione 1.3, «Sistemi distribuiti» affronta le diverse tecnologie disponibili per trarre il maggior beneficio dalla scalabilità orizzontale di Red Hat Enterprise Linuxl.

1.3. SISTEMI DISTRIBUITI

Per avere una scalabilità orizzontale Red Hat Enterprise Linux utilizza numerosi componenti di *elaborazione distribuita*. Le tecnologie che compongono una elaborazione distribuita sono suddivise in tre livelli:

Comunicazione

La scalabilità orizzontale richiede l'esecuzione simultanea di numerosi compiti (in parallelo). Per questo motivo i suddetti compiti devono essere *in comunicazione tra loro* per coordinare il lavoro. A tale scopo una piattaforma con scalabilità orizzontale dovrebbe essere in grado di condividere i compiti su sistemi multipli.

Storage

Per soddisfare i requisiti di scalabilità orizzontale non è sufficiente implementare uno storage tramite i dischi locali. È necesario usare uno storage condiviso o distribuito, uno con un livello di astrazione che permetta alla capacità di un volume di storage singolo, di aumentare con l'aggiunta di nuovo hardware di storage.

Gestione

Il copito più importante di una elaborazione distribuita è il livello di *gestione*. Questo livello coordina tutti i componenti hardware e software, gestisce in modo efficiente le comunicazioni, lo storage e l'uso di risorse condivise.

La seguente sezione descrive in modo più dettagliato le tecnologie all'interno di ogni livello.

1.3.1. Comunicazione

Il livello di Comunicazione assicura il trasporto dei dati ed è composto da due parti:

- Hardware
- Software

Il modo più semplice (e veloce) di comunicare per sistemi multipli è quello di utilizzare una *memoria condivisa*. Ciò comporta l'uso di operazioni di lettura/scrittura della memoria conosciute; una memoria condivisa possiede una larghezza di banda molto elevata, una bassa latenza ed un overhead basso di operazioni comuni di lettura/scrittura della memoria.

Ethernet

Il modo più comune per una comunicazione tra computer è attraverso Ethernet. Al giorno d'oggi *Gigabit Ethernet* (GbE) è presente sui sistemi per impostazione predefinita, e numerosi server includono 2-4 porte di Gigabit Ethernet. GbE fornisce una buona larghezza di banda e latenza, le quali risultano essere la base di numerosi sistemi distribuiti in uso. Anche in presenza di sistemi con hardware di rete più veloce, è ancora comune l'uso di GbE come interfaccia di gestione.

10GbE

Ten Gigabit Ethernet (10GbE) è attualmente in rapida crescita ed è sempre più accettato in server mid-range e high end. 10GbE fornisce una larghezza di banda dieci volte maggiore rispetto a GbE. Se siete in possesso di un processore multi-core moderno sarà possibile usufruire della possibilità di ripristinare l'equilibrio tra comunicazione e processazione. È possibile confrontare un sistema a core singolo usando GbE ad un sistema con otto core usando 10GbE. Usato in questo modo 10GbE è molto utile per il mantenimento delle prestazioni generali del sistema, riducendo eventuali limitazioni durante le comunicazioni.

Sfortunatamente 10GbE è costoso. Anche se il costo di 10GbE NIC è diminuito, il prezzo dei collegamenti (ed in particolare delle fibre ottiche) resta elevato, altresì gli interruttori di rete 10GbE sono estremamente costosi. Col tempo questi costi dovrebbero ridursi ma al giorno d'oggi 10GbE è ampiamente usato con applicazioni critiche per le prestazioni e come macchina principale di un locale server.

Infiniband

Infiniband offre prestazioni ancora più elevate di 10GbE. In aggiunta a collegamenti di rete UDP e TCP/IP usati con Ethernet, Infiniband supporta anche una comunicazione della memoria condivisa. Ciò permette a Infiniband di operare tra sistemi tramite un *remote direct memory access*(RDMA).

L'uso di RDMA permette a Infiniband di spostare i dati direttamente tra sistemi senza l'overhead delle connessioni socket o TCP/IP. Così facendo verrà ridotta la latenza, che risulta critica per alcune applicazioni.

Infiniband viene più comunemente usato con applicazioni *High Performance Technical Computing* (HPTC) che richiedono una larghezza di banda elevata, e una latenza e overhead più bassi. Molte applicazioni di supercomputing traggono beneficio da questa implementazione, al punto che il modo migliore per aumentare le prestazioni è quello di investire su Infiniband al posto di usare processori più veloci o più memoria.

RoCCE

RDMA over Ethernet (RoCCE) implementa comunicazioni simili a Infiniband (incluso RDMA) attraverso una infrastruttura 10GbE. A causa della riduzione dei costi associati con un aumento dei prodotti 10GbE, è possibile prevedere un uso più esteso di RDMA e RoCCE su un certo numero di sistemi e applicazioni.

Questi metodi di comunicazione sono completamente supportati da Red Hat per un loro utilizzo con Red Hat Enterprise Linux 6.

1.3.2. Storage

Un ambiente che utilizza una elaborazione distribuita usa istanze multiple di storage condiviso. Ciò può significare:

- Sistemi multipli che archiviano dati in una singola posizione
- Una unità di storage (es. un volume) composto da elementi di storage multipli

L'esempio più comune è l'unità disco locale montata su un sistema. Ciò risulta appropriato per operazioni IT dove tutte le applicazioni sono ospitate su un unico host o su un numero ristretto di host. Tuttavia, se l'infrastruttura presenta dozzine o anche centinaia di sistemi, la gestione di numerosi dischi di storage diviene un compito difficoltoso.

Lo storage distribuito aggiunge un livello in grado di facilitare e automatizzare l'amministrazione hardware dello storage con l'aumentare delle dimensioni. L'uso di sistemi multipli che condividono un certo numero di istanze di storage riduce il numero di dispositivi gestiti da un amministratore.

Il consolidamento delle capacità di archiviazione di elementi di storage multipli in un volume, è una operazione utile sia per gli utenti che per gli amministratori. Questo tipo di storage distribuito fornisce un livello di astrazione ai gruppi di storage: gli utenti vedono una unità singola di storage la quale può essere ampliata da un amministratore aggiungendo hardware. Alcune tecnologie che permettono di usare uno storage distribuito forniscono anche benefici aggiuntivi, come ad esempio il failover e il multipathing.

NFS

Network File System (NFS) permette a utenti o server di montare e usare la stessa istanza di storage remoto tramite TCP o UDP. NFS viene usato per contenere i dati condivisi da applicazioni multiple. Esso è conveniente per l'archiviazione di un numero molto grande di dati.

SAN

Storage Area Networks (SAN) utilizza un protocollo iSCSI o un Fibre Channel per fornire un accesso remoto allo storage. L'infrastruttura del Fibre Channel (come ad esempio gli adattatori host bus Fibre Channel, interruttori e storage array) uniscono una elevata prestazione, larghezza di banda e uno storage molto grande. Le SAN separano lo storage della processazione e forniscono una elevata flessibilità durante la creazione del sistema.

L'altro vantaggio molto importante delle SAN è quella di fornire un ambiente di gestione per eseguire compiti importanti di amministrazione hardware dello storage. Questi compiti includono:

- Controllo accesso allo storage
- Gestione di quantità molto grandi di dati
- Provisioning dei sistemi
- Backup e riproduzione dei dati
- Esecuzione di istantanee
- Supporto del failover dei sistemi
- Assicurazione integrità dei dati

- Migrazione dei dati

GFS2

Il file system *Global File System 2* (GFS2) di Red Hat fornisce diverse capacità specifiche. La funzione di base di GFS2 è quella di fornire un file system singolo, incluso un accesso lettura/scrittura simultaneo condiviso su membri multipli di un cluster. Ciò significa che ogni membro del cluster visualizza esattamente gli stessi dati "sul disco" nel file system GFS2.

GFS2 permette a tutti i sistemi di avere un accesso simultaneo al "disco". Per mantenere l'integrità dei dati, GFS2 utilizza un *Distributed Lock Manager* (DLM), il quale permette solo ad un sistema per volta di scrivere in una posizione specifica.

GFS2 è particolarmente ideale per applicazioni di failover che richiedono una elevata disponibilità di storage.

Per maggiori informazioni su GFS2 consultare il *Global File System 2*. Per maggiori informazioni sullo storage in generale consultare la *Storage Administration Guide*. Entrambe le guide sono disponibili su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

1.3.3. Reti convergenti

Normalmente la comunicazione sulla rete viene eseguita attraverso Ethernet, con il traffico di archiviazione che utilizza un ambiente Fibre Channel SAN apposito. È comune avere un link seriale o di rete per la gestione del sistema, e probabilmente anche un *heartbeat*[2]. Ne risulta la presenza di un singolo server su reti multiple.

Fornire connessioni multiple su ogni server risulta essere costoso, ingombrante e complesso da gestire. Per questo motivo è nata la necessità di avere tutte le connessioni incorporate all'interno di una unica connessione. *Fibre Channel over Ethernet* (FCoE) e *Internet SCSI* (iSCSI), risolvono questo problema.

FCoE

Con FCoE i comandi del fibre channel standard e i pacchetti dati vengono trasportati attraverso una infrastruttura fisica di 10GbE tramite un *converged network card* (CNA) singolo. Il traffico ethernet TCP/IP standard e le operazioni di storage del fibre channel possono essere trasportati usando lo stesso link. FCoE utilizza una scheda dell'interfaccia di rete fisica (ed un cavo) per connessioni storage/rete logiche multiple

FCoE offre i seguenti vantaggi:

Numero ridotto di connessioni

FCoE riduce della metà i numeri delle connessioni di rete ad un server. Per ragioni di disponibilità e prestazioni è ancora possibile selezionare connessioni multiple; tuttavia una singola connessione è in grado di fornire sia una connettività di rete che una di storage. Questa impostazione è specialmente utile per server pizza box e blade poichè entrambi presentano uno spazio limitato per i componenti.

Costo più basso

Una riduzione immediata delle connessioni significa ridurre anche il numero di cavi, interruttori e altri strumenti di rete. Ethernel dispone anche di una vasta gamma di dispositivi; il costo delle reti diminuisce drammaticamente a causa di un aumento del numero da milioni a miliardi di dispositivi, una conferma di quanto detto la si ha nella riduzione del costo dei dispositivi Ethernet 100Mb e gigabit.

In modo simile 10GbE saranno meno costosi in futuro a causa di un loro maggior uso. Con implementazioni sempre più numerose di hardware CNA con singoli chip, sarà possibile aumentarne il loro volume nel mercato, riducendone così i costi.

iSCSI

Internet SCSI (iSCSI) è un altro tipo di protocollo di rete convergente; Esso rappresenta una alternativa a FCoE. Come il fibre channel, iSCSI fornisce uno storage di livello del blocco attraverso una rete. Tuttavia iSCSI non fornisce un ambiente di gestione completo. Il vantaggio principale di iSCSI rispetto a FCoE è quello di fornire gran parte delle capacità e flessibilità di fibre channel, ma a costi più bassi.

---

[1] Red Hat Certified Engineer. Per maggiori informazioni consultate http://www.redhat.com/training/certifications/rhce/.

[2] Un *Heartbeat* rappresenta lo scambio di messaggi tra sistemi per assicurare un funzionamento corretto. Se un sistema "perde heartbeat" viene considerato "fallito" e quindi arrestato. A questo punto un secondo sistema subentra al sistema fallito prendendosi carico dei suoi compiti.

CAPITOLO 2. FUNZIONI PER LE PRESTAZIONI DI RED HAT ENTERPRISE LINUX 6

2.1. SUPPORTO 64-BIT.

Red Hat Enterprise Linux 6 supporta processori a 64-bit; i suddetti processori possono usare in teoria fino a 16 *exabyte* di memoria. Per disponibilità generale (GA), Red Hat Enterprise Linux 6 è testato e certificato per supportare fino a 8TB di memoria fisica.

È previsto un aumento della dimensione supportata da Red Hat Enterprise Linux 6 attraverso aggiornamenti futuri, questo poichè Red Hat introduce costantemente miglioramenti relativi alle funzioni che agevolano l'uso di blocchi di memoria più grandi. Esempi di tali miglioramenti (a partire da Red Hat Enterprise Linux 6 GA) sono:

- Huge pages e transparent huge pages
- Miglioramenti Non-Uniform Memory Access (NUMA)

Questi miglioramenti vengono riportati in maggior dettaglio nelle sezioni seguenti.

Huge pages e transparent huge pages

L'implementazione delle *huge pages* con Red Hat Enterprise Linux 6 permette al sistema di gestire l'uso della memoria in modo efficiente in presenza di diversi carichi di lavoro. Le Huge pages utilizzano dinamicamente pagine di 2 MB rispetto allo standard di 4 KB, permettendo alle applicazioni di scalare correttamente durante la processazione dei gigabyte e terabyte di memoria.

Le Huge pages sono difficili da creare, gestire ed usare manualmente. Per ovviare a questo problema Red Hat Enterprise 6 dispone delle *transparent huge pages* (THP). THP gestisce automaticamente numerosi processi presenti nell'uso delle huge pages.

Per maggiori informazioni sulle huge pages e THP consultare Sezione 5.2, «Huge pages e transparent huge pages».

Miglioramenti NUMA

Numerosi sistemi moderni supportano ora il *Non-Uniform Memory Access* (NUMA). NUMA semplifica il design e la creazione di hardware di sistemi molto grandi; tuttavia esso aggiunge un livello di complessità allo sviluppo delle applicazioni. Per esempio, NUMA implementa sia la memoria locale che quella remota, quest'ultima impiega un periodo più lungo per l'accesso rispetto alla memoria locale. Questa funzione (insieme ad altre) presenta alcune problematiche relative alle prestazioni dei sistemi operativi, applicazioni e delle configurazioni del sistema.

Red Hat Enterprise Linux 6 è più idoneo per NUMA a causa di nuove funzioni che assistono durante la gestione di applicazioni e utenti sui sistemi NUMA. Queste funzioni includono CPU affinity, CPU pinning (cpusets), numactl ed i gruppi di controllo, e permettono così ad un processo (affinity) o applicazione (pinning) di associarsi ad una CPU o insieme di CPU specifiche.

Per maggiori informazioni sul supporto NUMA con Red Hat Enterprise Linux 6 consultare Sezione 4.1.1, «Tipologia NUMA e CPU».

2.2. TICKET SPINLOCKS

Molta importanza in qualsiasi design è quello di assicurare che un processo non alteri la memoria usata da un altro processo. Modifiche non controllate dei dati in memoria possono risultare in una loro corruzione ed eventuale crash del sistema. Per impedire il verificarsi di queste situazioni il sistema

operativo permette ad un processo di bloccare una sezione di memoria, eseguire una operazione e sblocare o "liberare" la sezione interessata.

Una implementazione comune del blocco della memoria viene eseguita tramite *spin locks*, i quali permettono ad un processo di controllare la disponibilità di un blocco e prenderlo appena quest'ultimo risulta disponibile. In presenza di processi multipli in competizione tra loro per lo stesso blocco, il primo processo che esegue la richiesta otterrà l'uso del blocco. Quando tutti i processi hanno lo stesso accesso alla memoria, questo tipo di impostazione risulta essere la più "imparziale".

Sfortunatamente su un sistema NUMA non tutti i processi hanno un accesso simile ai blocchi. I processi presenti sullo stesso nodo NUMA del blocco, hanno un vantaggio nell'ottenere il blocco rispetto ad altri processi. I processi su nodi NUMA remoti spesso presentano un lock starvation il quale impatta negativamente sulle rispettive prestazioni.

Per risolvere questo problema Red Hat Enterprise Linux implementa i *ticket spinlocks*. Questa funzione apporta un meccanismo di prenotazione per il blocco, e permette a *tutti* i processi di ottenere il blocco seguendo un ordine basato sulla cronologia delle richieste. Questa impostazione elimina il problema relativo ai tempi della richiesta e di imparzialità.

Anche se un ticket spinlock presenta un overhead maggiore rispetto ad un spinlock ordinario, esso risulta essere quello più idoneo nei sistemi NUMA.

2.3. STRUTTURA ELENCO DINAMICO

Il sistema operativo ha bisogno di un insieme di informazioni su ogni processore del sistema. Con Red Hat Enterprise Linux 5 questo set di informazioni veniva assegnato ad un array di dimensione fissa nella memoria. Le informazioni presenti su ogni processore venivano acquisite eseguendo una indicizzazione nel suddetto array. Questo metodo era veloce e semplice per i sistemi con pochi processori.

Tuttavia con l'aumentare del numero di processori nel sistema questo metodo genera un overhead molto elevato. Poichè l'array di dimensione fissa è una risorsa singola e condivisa, tale caratteristica può rappresentare una limitazione poichè il numero di processori che richiede un accesso aumenta col tempo.

Per risolvere questo problema Red Hat Enterprise Linux 6 utilizza una *struttura dell'elenco dinamico* per le informazioni del processore. Ciò permette una assegnazione dinamica dell'array usato per le informazioni del processore: in presenza di otto processori nel sistema, verranno create nell'elenco solo otto voci. In presenza di 2048 processori allora verranno create 2048 voci.

Una struttura dell'elenco dinamico permette un blocco di una porzione più piccola. Per esempio, se è necessario aggiornare contemporaneamente alcune informazioni per processori 6, 72, 183, 657, 931 e 1546, tale operazione può essere svolta con maggiore parallelismo. Situazioni come queste si verificano più frequentemente su sistemi più grandi ad elevate prestazioni rispetto a sistemi più piccoli.

2.4. TICKLESS KERNEL

Nelle precedenti versioni di Red Hat Enterprise Linux il kernel utilizzava un meccanismo basato sul timer in grado di generare continuamente un interrupt del sistema. Durante ogni interrupt il sistema esegue una *verifica*; e cioè controlla la presenza di eventuali eventi da eseguire.

In base alle impostazioni l'interruzione o il *timer tick* si può verificare centinaia o migliaia di volte al secondo, senza considerare il carico di lavoro del sistema. Su un sistema con un carico di lavoro non molto elevato questo comportamento può impattare negativamente sul *consumo energetico*, impedendo al processore di entrare in uno stato di sospensione "sleep mode". Il sistema utilizza il livello minimo di alimentazione se risulta essere sospeso.

Il modo più efficiente di utilizzare l'alimentazione per un sistema è quello di eseguire operazioni il più velocemente possibile, entrare in uno stato di sospensione e restarci il più a lungo possibile. Per fare ciò Red Hat Enterprise Linux 6 utilizza un *tickless kernel*. Con esso il timer delle interruzioni è stato rimosso dall' idle loop, trasformando così Red Hat Enterprise Linux 6 in un ambiente basato sugli interrupt.

Il tickless kernel permette ad un sistema di entrare in uno stato di sospensione durante periodi di inattività "idle" e di riattivarsi velocemente in presenza di eventi da eseguire.

Per maggiori informazioni consultate la *Power Management Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

2.5. GRUPPI DI CONTROLLO

Red Hat Enterprise Linux fornisce un certo numero di opzioni utili per l'ottimizzazione delle prestazioni. Sistemi molto grandi, con centinaia di processori, possono essere regolati in modo da avere una prestazione ottimale. Per regolare questi sistemi è necessario avere una conoscenza approfondita ed un carico di lavoro definito. In passato quando i suddetti sistemi erano costosi e non molto numerosi era normale avere un trattamento speciale nei loro confronti, col tempo, e con l'aumentare dei suddetti sistemi fino a diventare "normali", è aumentata la necessità di utilizzare strumenti più efficaci.

Al giorno d'oggi per la consolidazione dei servizi vengono utilizzati sistemi più potenti. Carichi di lavoro eseguiti da un minimo di quattro ad un massimo di otto server, vengono ora eseguiti su un singolo server. Come affrontato precedentemente in Sezione 1.2.1, «Elaborazione parallela», attualmente i sistemi di livello medio contengono un numero maggiore di core rispetto a sistemi passati ad elevate prestazioni.

Attualmente numerose applicazioni vengono create per una processazione parallela tramite thread multipli o processi di miglioramento delle prestazioni. Tuttavia solo poche applicazioni possono usare in modo più effettivo più di otto thread. Per questo motivo installare applicazioni multiple su sistemi a 32-CPU per massimizzare la capacità.

Considerate che sistemi mainstream poco costosi e più piccoli sono oggi ad un livello simile a quello di macchine ad elevate prestazioni e molto costose usate in passato. Macchine ad elevate prestazioni meno costose conferivano la possibilità agli architetti di sistema di consolidare più servizi con un numero più ristretto di macchine.

Tuttavia alcune risorse (come ad esempio I/O e comunicazioni di rete) sono condivise e non aumentano in modo simile al conteggio della CPU. Per questo motivo un sistema che presenta un certo numero di applicazioni può avere una diminuzione delle prestazioni quando una applicazione "monopolizza" per tempi molto estesi una singola risorsa.

Per risolvere questo problema Red Hat Enterprise Linux 6 supporta ora i *control groups* (cgroups). Cgroups permettono ad un amministratore di assegnare le risorse a compiti specifici. Ciò significa assegnare 80% di quattro CPU, 60GB di memoria e 40% di I/O del disco ad una applicazione del database. Una applicazione web in esecuzione sullo stesso ssistema può ricevere due CPU, 2 GB di memoria e 50% di larghezza di banda disponibile della rete.

Ne risulta una migliore prestazione poichè il sistema impedisce un consumo eccessivo di risorse sia da parte del database che delle applicazioni web. Altresì numerosi aspetti di cgroups sono *self-tuning*, e permettono al sistema di rispondere in modo adeguato al variare del carico di lavoro.

Un cgroup presenta due componenti principali:

- Un elenco di compiti assegnati al cgroup

- Risorse assegnate ai suddetti compiti

I compiti assegnati al cgroup vengono eseguiti *all'interno* del cgroup. Ciò permetterà ad un amministratore di gestire una intera applicazione come singola unità. L'amministratore sarà in grado di configurare le assegnazioni per le seguenti risorse:

- CPUsets
- Memoria
- I/O
- Rete (larghezza di banda)

All'interno dei CPUset i cgroup permettono agli amministratori di configurare il numero di CPU, l'affinità per specifiche CPU o nodi [3], e la quantità di tempo della CPU usata da un insieme di compiti. L'uso di cgroup per configurare i CPUset risulta essere vitale per assicurare un buon livello di prestazione, ed impedisce il consumo eccessivo di risorse da parte di una applicazione, assicurando al tempo stesso un livello accettabile di risorse per il tempo di CPU.

La larghezza di banda I/O e di rete sono gestite da altri controllori di risorse. I suddetti controllori permettono di determinare la quantità di larghezza di banda consumata dai compiti all'interno di un cgroup, e assicurano al tempo stesso che i compiti presenti al suo interno non consumino un livello eccessivo o che abbiano un livello troppo basso di risorse.

I Cgroup permettono ad un amministratore di definire e assegnare, con un livello molto alto, le risorse necessarie alle varie applicazioni. Il sistema gestisce automaticamente ed in modo equilibrato le varie applicazioni, rendendo disponibile un buon livello prevedibile di prestazioni e ottimizzando il sistema in generale.

Per maggiori informazioni su come usare i control groups consultare la *Resource Management Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

2.6. MIGLIORAMENTI DEL FILE SYSTEM E STORAGE

Red Hat Enterprise Linux 6 introduce numerosi miglioramenti alla gestione del file system e dello storage. Alcuni dei miglioramenti più importanti includono il supporto XFS e ext4. Per informazioni più dettagliate sui miglioramenti introdotti consultare Capitolo 7, *File System*.

Ext4

Ext4 è il file system predefinito di Red Hat Enterprise Linux 6 e risulta essere la quarta generazione della famiglia di file system EXT. Grazie alla sua introduzione è ora possibile avere una dimensione teorica massima del file system pari a 1 exabyte (non supportata da Red Hat), e una dimensione massima per un file singolo di 16TB. Red Hat Enterprise Linux 6 supporta una dimensione massima del file system pari a 16TB ed una dimensione massima per file singolo di 16TB. Insieme ad una capacità di storage molto più grande, ext4 include anche diverse nuove funzioni come ad esempio:

- Metadati basati sulle estensioni
- Assegnazione ritardata
- Journal check-summing

Per maggiori informazioni sul file system ext4 consultare Sezione 7.3.1, «File system Ext4».

XFS

XFS è un file system con journal maturo e robusto in grado di supportare file system e file molto grandi su un singolo host. Questo file system è stato originariamente creato da SGI ed è stato usato in modo esteso su server molto grandi e storage array. Le funzioni di XFS includono:

- Assegnazione ritardata
- Inode assegnati dinamicamente
- indicizzazione B-tree per una scalabilità della gestione di spazio disponibile
- Aumento del file system e deframmentazione online
- Algoritmi read-ahead per metadati sofisticati

Anche se XFS raggiunge un ordine di grandezza exabyte, la dimensione massima di un fle system XFS supportata da Red Hat è 100TB. Per maggiori informazioni su XFS consultare Sezione 7.3.2, «Il file system XFS».

Unità d'avvio molto grandi

Un BIOS tradizionale supporta una dimensione massima del disco pari a 2.2TB. I sistemi Red Hat Enterprise Linux 6 che utilizzano un BIOS sono in grado di supportare dischi più grandi di 2.2TB, attraverso l'utilizzo di una nuova struttura chiamata *Global Partition Table* (GPT). GPT può essere usato solo per dischi dati e non può essere usato per unità d'avvio con BIOS; per questo motivo le suddette unità possono avere una dimensione massima di 2.2TB. Il BIOS è stato creato originariamente da IBM PC; mentre il BIOS si è evoluto in modo da essere adattato all'hardware moderno, *Unified Extensible Firmware Interface* (UEFI) è stato creato per supportare hardware emergente.

Red Hat Enterprise Linux 6 supporta anche la UEFI, usata per sostituire il BIOS (ancora supportato). Su sistemi con UEFI e Red Hat Enterprise Linux 6 è possibile usare partizioni di 2.2TB e GPT (e dimensioni più grandi) sia per partizioni dati che per boot.

IMPORTANTE

Red Hat Enterprise Linux 6 non supporta UEFI per sistemi x86 a 32-bit.

IMPORTANTE

Da notare che le configurazioni d'avvio di UEFI e BIOS differiscono in modo significativo. Per questo motivo il sistema installato deve essere avviato usando lo stesso firmware usato durante l'installazione. Non sarà possibile installare il sistema operativo su di un sistema che utilizza il BIOS per poi avviare questa installazione su di un sistema che utilizza UEFI.

Red Hat Enterprise Linux 6 supporta la versione 2.2 delle specifiche UEFI. Hardware che supportano la versione 2.3 delle specifiche UEFI o più recenti, possono essere avviati ed operare con Red Hat Enterprise Linux 6 ma le funzionalità aggiuntive definite dalle specifiche non saranno disponibili. Le specifiche UEFI sono disponibili su http://www.uefi.org/specs/agreement/.

[3] Generalmente un nodo viene definito come set di CPU o core all'interno di un socket.

CAPITOLO 3. MONITORAGGIO ED ANALISI DELLE PRESTAZIONI DEL SISTEMA

Questo capitolo introduce brevemente gli strumenti usati per monitorare ed analizzare le prestazioni delle applicazioni e del sistema, riporta altresì le situazioni nelle quali ogni strumento risulta essere il più idoneo. I dati raccolti per ogni strumento possono evidenziare possibili limiti o altre problematiche che contribuiscono ad una prestazioni non ottimale.

3.1. FILE SYSTEM PROC

Il file system **proc** è una directory che contiene una gerarchia di file che rappresentano lo stato corrente del kernel di Linux. Esso permette alle applicazioni e agli utenti di visualizzare una panoramica del sistema del kernel.

La directory **proc** contiene anche le informazioni relative all'hardware del sistema e di qualsiasi processo attualmente in esecuzione. La maggior parte di questi file sono di sola-lettura, ma alcuni file (principalmente quelli in **/proc/sys**) possono essere manipolati da utenti e da applicazioni, per la comunicazione al kernel delle modifiche sulla configurazione.

Per maggiori informazioni sulla visualizzazione e modifica dei file nella directory **proc** consultare la *Deployment Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.2. GNOME E KDE SYSTEM MONITORS

Gli ambienti KDE e GNOME presentano entrambi strumenti grafici per un ausilio al monitoraggio e modifica del comportamento del sistema.

GNOME System Monitor

Il **GNOME System Monitor** mostra informazioni di base e permette di monitorare l'uso dei processi, risorse e file system del sistema. Apritelo usando il comando **gnome-system-monitor** nel **Terminal**, o selezionare nel menu **Applicazioni** e **Strumenti del sistema** > **Monitor del sistema**.

GNOME System Monitor presenta quattro schede:

Sistema

Mostra le informazioni di base sul software e hardware del computer.

Processi

Mostra i processi attivi e il rapporto tra i suddetti processi insieme alle informazioni dettagliate di ogni processo. Permette altresì di filtrare i processi ed eseguire determinate azioni (avvio,arresto, kill, modifica della priorità ecc.).

Risorse

Mostra l'uso del tempo della CPU corrente, l'uso della memoria e dello spazio di swap e della rete.

File System

Elenca tutti i file system montati insieme alle informazioni di base su ognuno di essi, come ad esempio il tipo di file system, il mount point e l'uso della memoria.

Per maggiori informazioni su **GNOME System Monitor** consultare il menu **Aiuto** presente

nell'applicazione o la *Deployment Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

KDE System Guard

KDE System Guard permette di controllare i processi in esecuzione ed il carico del sistema corrente. Permette altresì di eseguire azioni nei confronti dei processi. A tale scopo usate il comando **ksysguard** nel **Terminal**, o selezionate **Lancia applicazioni** e successivamente **Applicazioni** > **Sistema** > **Controlla sistema**.

Sono disponibili due schede per il **KDE System Guard**:

Tabella processo

Per impostazione predefinita mostra un elenco di tutti i processi in esecuzione in ordine alfabetico. È possibile ordinare i processi prendendo in considerazione un certo numero di fattori, incluso l'uso della CPU totale, utilizzo della memoria condivisa o fisica, proprietario, e priorità. È possibile altresì filtrare i risultati visibili, eseguire una ricerca per processi specifici o eseguire determinate azioni su un processo.

Carico del sistema

Mostra una cronologia grafica sull'utilizzo della CPU, sull'uso dello spazio di swap, della memoria e della rete. Posizionate il mouse sopra i grafici per una analisi dettagliata.

Per maggiori informazioni su **KDE System Guard**, consultare il menu **Aiuto** all'interno dell'applicazione.

3.3. STRUMENTI DI MONITORAGGIO DELLA LINEA DI COMANDO INTERNI

In aggiunta agli strumenti di monitoraggio grafico, Red Hat Enterprise Linux fornisce numerosi strumenti per il controllo di un sistema dalla linea di comando. Il vantaggio di questi strumenti è quello di poter utilizzarli esternamente al runlevel 5. Questa sezione affronta brevemente ogni strumento e il loro uso migliore.

`top`

top fornisce una panoramica dinamica in tempo reale dei processi in esecuzione sul sistema. Esso è in grado di mostrare una varietà di informazioni incluso il sommario del sistema, i compiti gestiti dal kernel di Linux e presenta una limitata abilità di manipolare i processi. Sia le informazioni che le operazioni riportate sono altamente configurabili, e qualsiasi informazione sulla configurazione può essere resa persistente durante i riavvii.

Per impostazione predefinita i processi mostrati sono ordinati in base alla percentuale di utilizzo della CPU, fornendo così un resoconto semplice da consultare sui processi che utilizzano il numero maggiore di risorse.

Per informazioni dettagliate su come usare **top** consultare la pagina man relativa: **`man top`**.

`ps`

ps esegue una istantanea di un gruppo selezionato di processi attivi. Per impostazione predefinita questo gruppo è limitato ai processi posseduti dall'utente corrente ed associato con lo stesso terminale.

Esso può fornire informazioni più dettagliate sui processi rispetto a **top** ma non è dinamico.

Per informazioni dettagliate su come usare **ps** consultare la pagina man relativa: **man ps**.

vmstat

vmstat (Virtual Memory Statistics) esegue l'output di notifiche istantanee sulla memoria, paging, I/O del blocco, interrupt, attività della CPU e processi del sistema.

Anche se non dinamico come **top** è possibile specificare un intervallo di campionamento il quale permette di osservare l'attività del sistema quasi in tempo reale.

Per informazioni dettagliate su come usare **vmstat** consultare la pagina man relativa: **man vmstat**.

sar

sar (System Activity Reporter) raccoglie e riporta le informazioni sull'attività del sistema giornaliera. L'output predefinito riporta l'uso giornaliero della CPU ad un intervallo di dieci minuti dall'inizio del giorno:

```
12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal
%idle
12:10:01 AM     all      0.10      0.00      0.15      2.96      0.00
96.79
12:20:01 AM     all      0.09      0.00      0.13      3.16      0.00
96.61
12:30:01 AM     all      0.09      0.00      0.14      2.11      0.00
97.66
...
```

Questo strumento rappresenta una alternativa utile al tentativo di creare notifiche periodiche sull'attività del sistema attraverso **top** o strumenti simili.

Per informazioni dettagliate su come usare **sar** consultare la pagina man relativa: **man sar**.

3.4. TUNED E KTUNE

Tuned è un demone in grado di controllare e acquisire i dati sull'utilizzo dei componenti del sistema, e ottimizza dinamicamente le impostazioni del sistema in base alle informazioni in possesso. Esso reagisce alle modifiche relative all'uso della CPU e della rete e modifica le impostazioni per migliorare le prestazioni in dispositivi attivi o riduce il consumo energetico in quelli inattivi.

ktune, usato insieme a **tuned-adm**, fornisce un numero di profili di ottimizzazione preconfigurati per il miglioramento delle prestazioni e la riduzione del consumo energetico durante le diverse implementazioni. Creare o modificare nuovi profili per soluzioni nuove adatte all'ambiente desiderato.

I profili inclusi come parte di **tuned-adm** includono:

default

Il profilo di risparmio energetico predefinito. Questo è il profilo di base e abilita solo i plug-in CPU e del disco. Da notare che questo non è simile alla disabilitazione di **tuned-adm** dove sia **tuned** che **ktune** sono disabilitati.

latency-performance

Un profilo server per l'ottimizzazione delle prestazioni della latenza. Disabilita i maccanismi di risparmio energetico **tuned** e **ktune**. La modalità **cpuspeed** varia in **performance**. Per ogni dispositivo l'I/O elevator viene modificato in **deadline**. Per la qualità del servizio di gestione

dell'alimentazione, viene registrato il valore **0** di **cpu_dma_latency**.

throughput-performance

Un profilo server per l'ottimizzazione tipica delle prestazioni di output netto. Questo profilo è consigliato se il sistema non presenta alcuno storage di classe enterprise. Simile a **latency-performance** ad eccezione:

- ***kernel.sched_min_granularity_ns*** (scheduler minimal preemption granularity) è impostato su **10** millisecondi,
- ***kernel.sched_wakeup_granularity_ns*** (scheduler wake-up granularity) impostato su **15** millisecondi,
- ***vm.dirty_ratio*** (virtual machine dirty ratio) impostato su 40%, e
- le transparent huge pages sono abilitate.

enterprise-storage

Questo profilo è consigliato per configurazioni del server con dimensioni enterprise con uno storage di classe enterprise, inclusa una protezione della cache del controller con un backup tramite batteria ed una gestione della cache su-disco. Simile al profilo **throughput-performance**, con una eccezione: i file system sono rimontati con **barrier=0**.

virtual-guest

Questo profilo è consigliato per configurazioni del server di dimensioni enterprise con uno storage di classe enterprise, inclusa una protezione della cache del controller con un backup tramite batteria ed una gestione della cache su-disco. Simile al profilo **throughput-performance**, ad eccezione:

- ***readahead*** valore impostato su **4x**, e
- file system non root/boot rimontati con **barrier=0**.

virtual-host

In base al profilo ***enterprise-storage***, ***virtual-host*** diminuisce anche la tendenza allo swap della memoria virtuale e abilita un writeback più aggressivo di pagine non ancora salvate. Questo profilo è disponibile in Red Hat Enterprise Linux 6.3 e versioni più recenti, ed è il profilo consigliato per gli host di virtualizzazione, incluso gli host di KVM e Red Hat Enterprise Virtualization.

Consultare Red Hat Enterprise Linux 6 *Power Management Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/, per maggiori informazioni su **tuned** e **ktune**.

3.5. PROFILER DELL'APPLICAZIONE

Il profiling è quel processo di raccolta delle informazioni sui comportamenti del programma durante la sua esecuzione. Analizzando una applicazione sarà possibile determinare le aree di un programma da ottimizzare per poter aumentare la velocità generale del programma stesso, ridurre l'uso della memoria ecc. Gli strumenti per il profiling dell'applicazione aiutano a semplificare questo processo.

Con Red Hat Enterprise Linux 6 è possibile utilizzare i seguenti tipi di strumenti: **SystemTap**, **OProfile**

e **Valgrind**. La documentazione dei suddetti strumenti va oltre lo scopo di questa guida; tuttavia questa sezione fornisce i link per ottenere informazioni aggiuntive ed una breve panoramica sui compiti con i quali usare il profiler più idoneo.

3.5.1. SystemTap

SystemTap è uno strumento di rilevamento e monitoraggio che permette agli utenti di studiare e monitorare le attività del sistema operativo (in particolare del kernel) in modo dettagliato. Esso fornisce informazioni simili all'output di strumenti come **netstat**, **ps**, **top**, e **iostat**, ma include un numero maggiore di opzioni d'analisi e di filtraggio per le informazioni raccolte.

SystemTap fornisce una analisi più precisa e dettagliata sulle attività e sul comportamento delle applicazioni del sistema in modo da individuare con accuratezza possibili problematiche.

La funzione Callgraph plug-in per Eclipse utilizza SystemTap come backend, e permette un controllo dello stato di un programma, incluse le invocazioni della funzione, ritorni, i tempi e le variabili spazio-utente, queste informazioni vengono rese visibili per una ottimizzazione più semplice.

Per maggiori informazioni su SystemTap, consultare la *SystemTap Beginners Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.5.2. OProfile

OProfile (**oprofile**) è un tool di controllo delle prestazioni dell'intero sistema. Il suddetto tool usa un hardware di controllo delle prestazioni sul processore, per ottenere informazioni sul kernel e sugli eseguibili presenti sul sistema, come ad esempio quando ci si riferisce alla memoria, al numero delle richieste della cache L2 e al numero degli interrupt hardware ricevuto. Può essere usato per determinare l'uso del processore e quali applicazioni e servizi maggiormente utilizzati.

OProfile può essere usato anche con Eclipse tramite Eclipse OProfile plug-in. Questo plug-in permette agli utenti di determinare facilmente le aree che consumano una quantità di tempo maggiore del proprio codice, ed eseguire tutte le funzioni della linea di comando di OProfile con una visualizzazione dettagliata dei risultati.

Tuttavia gli utenti devono tener presente alcune limitazioni di OProfile:

- Gli esempi di monitoraggio delle prestazioni potrebbero non essere precisi - poichè il processore può eseguire le istruzioni in ordine sparso, un esempio potrebbe essere registrato da una qualsiasi altra istruzione vicina, e non utilizzare l'istruzione che ha innescato l'interrupt.

- Poichè OProfile può essere usato dall'intero sistema e al tempo stesso prevede avvii e arresti multipli dei processi, sarà possibile archiviare un certo numero di esecuzioni. A tale scopo sarà necessario rimuovere i dati d'esempio relativi alle esecuzioni precedenti.

- Utilizzato principalmente per identificare i problemi con i processi CPU-limitati, e per questo motivo non è in grado di identificare i processi sospesi "sleeping" in attesa di blocchi per altri eventi.

Per maggiori informazioni sull'uso di OProfile consultate la *Deployment Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/, o la documentazione di **oprofile** presente sul sistema disponibile in `/usr/share/doc/oprofile-<version>`.

3.5.3. Valgrind

Valgrind rende disponibili un certo numero di strumenti per il profiling e di rilevamento per il miglioramento delle prestazioni e per una esecuzione corretta delle applicazione. Questi strumenti

sono in grado di rilevare errori relativi ai thread e alla memoria, insieme agli array overrun e dello stack, permettendo così una identificazione e correzione più semplice degli errori nel codice dell'applicazione. Essi sono in grado altresì di analizzare la cache, heap e la previsione del branch, per identificare i fattori in grado di aumentare la velocità dell'applicazione e ridurre l'uso della memoria.

Valgrind analizza l'applicazione eseguendola su una CPU sintetica, monitorandola e misurando le prestazioni del codice esistente durante la sua esecuzione. Successivamente stampa i "commenti" identificando ogni processo presente nell'esecuzione dell'applicazione su un descrittore del file specificato dall'utente, file, o socket di rete. Il livello di strumentazione varia in base allo strumento di Valgrind usato, e alle impostazioni, ma è importante notare che l'esecuzione di un codice "instrumentato" può richiedere un periodo dai 4-50 volte più lungo rispetto al periodo di esecuzione normale.

Valgrind può essere usato sull'applicazione senza alcuna ricompilazione aggiuntiva. Tuttavia poichè Valgrind utilizza informazioni di debugging per definire le problematiche nel codice, se l'applicazione e le librerie di supporto non sono state compilate con informazioni di debugging abilitate, è fortemente consigliato eseguire un nuovo processo di compilazione per includere queste informazioni.

Con Red Hat Enterprise Linux 6.4, Valgrind viene integrato con gdb (GNU Project Debugger) per migliorare l'efficienza del debugging.

Per maggiori informazioni su Valgrind consultare la *Developer Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/, o usando il comando **man valgrind** se avete installato il pacchetto valgrind. Ulteriore documentazione è disponibile anche in:

- **/usr/share/doc/valgrind-<version>/valgrind_manual.pdf**
- **/usr/share/doc/valgrind-<version>/html/index.html**

Per maggiori informazioni su come usare Valgrind per analizzare la memoria del sistema consultare Sezione 5.3, «Come usare Valgrind per una analisi sull'uso della memoria» .

3.5.4. Perf

perf rende disponibile un numero di contatori delle prestazioni utili e conferisce all'utente la possibilità di valutare l'impatto di altri comandi sul sistema:

perf stat

Fornisce statistiche generali per eventi normali delle prestazioni, incluso le istruzioni eseguite ed i cicli di orologio consumati. È possibile usare altresì i flag dell'opzione per ottenere i dati sugli eventi diversi da quelli di misurazione predefiniti. Con Red Hat Enterprise Linux 6.4 è possibile utilizzare **perf stat** per filtrare il monitoraggio in base a uno o più control groups (cgroups). Per maggiori informazioni consultare la pagina man: **man perf-stat**.

perf record

Registra i dati sulle prestazioni in un file analizzabile tramite **perf report**. Per maggiori informazioni consultare la pagina man: **man perf-record**.

perf report

Consulta i dati delle prestazioni da un file e analizza i dati registrati. Per maggiori informazioni consultare la pagina man: **man perf-report**.

perf list

Elenca gli eventi disponibili su una macchina specifica. Questi eventi varieranno in base alla configurazione software e all'hardware di monitoraggio delle prestazioni del sistema. Per maggiori informazioni consultare la pagina man: `man perf-list`.

`perf top`

Questo comando esegue una funzione simile a **top**. Esso è in grado di generare e visualizzare un profilo del contatore delle prestazioni in tempo reale. Per maggiori informazioni consultare la pagina man: `man perf-top`.

Maggiori informazioni su **perf** sono disponibili nella Red Hat Enterprise Linux *Developer Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.6. RED HAT ENTERPRISE MRG

Il componente Realtime di Red Hat Enterprise MRG include **Tuna**, uno strumento che permette agli utenti di modificare i valori regolabili del sistema e visualizzare i risultati delle modifiche. Sviluppato per un suo utilizzo con il componente Reltime, questo strumento può essere usato per ottimizzare i sistemi standard di Red Hat Enterprise Linux.

Con Tuna è possibile modificare o disabilitare attività non necessarie del sistema, incluso:

- Parametri BIOS relativi alla gestione energetica, rilevamento dell'errore e interruzioni per la gestione del sistema;
- Impostazioni della rete, come la funzione di ricezione dell'interruzione e l'uso di TCP;
- Attività del journaling nei file system
- Registrazione del sistema;
- Gestione delle interruzioni e dei processi dell'utente da parte di CPU specifiche o gruppo di CPU;
- Utilizzo dello spazio di swap, e
- Come comportarsi in presenza di eccezioni di uno stato out-of-memory.

Per maggiori informazioni sull'ottimizzazione di Red Hat Enterprise MRG con l'interfaccia Tuna consultare il capitolo "Regolazione generale del sistema" della *Realtime Tuning Guide*. Per informazioni dettagliate sull'utilizzo dell'interfaccia Tuna consultare la *Tuna User Guide*. Entrambe le guide sono disponibili su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_MRG/.

CAPITOLO 4. CPU

CPU è l'acronimo di *central processing unit*, tale nome non risulta essere totalmente corretto per la maggior parte dei sistemi poichè *centrale* implica *singolo*, e numerosi sistemi moderni possiedono più di una unità di processazione, o core. Fisicamente le CPU sono contenute in un pacchetto assegnato ad una scheda madre in un *socket*. Ogni socket presenta diverse connessioni: ad altri socket della CPU, controllori della memoria, controllori degli interrupt ed altri dispositivi periferici. Un socket per un sistema operativo è un raggruppamento logico di CPU e risorse associate. Questo concetto è importante in numerose discussioni sulla ottimizzazione della CPU.

Red Hat Enterprise Linux detiene un certo numero di informazioni sulle statistiche relative agli eventi della CPU del sistema; queste statistiche sono utili nella pianificazione di una strategia per il miglioramento delle prestazioni di una CPU. Sezione 4.1.2, «Ottimizzazione delle prestazioni della CPU» approfondisce alcune statistiche utili, dove trovarle e come analizzarle.

TIPOLIGIA

I computer più vecchi presentavano un numero di CPU per sistema più basso, permettendo così di implementare una architettura *Symmetric Multi-Processor* (SMP). Ogni CPU nel sistema aveva un accesso simile (o simmetrico) per la memoria disponibile. Col tempo il conteggio-per-socket della CPU è aumentato al punto che il tentativo di dare un accesso simmetrico a tutta la RAM nel sistema, è diventato un processo molto costoso. In tempi più recenti i sistemi con un conteggio CPU molto elevato implementano una architettura *Non-Uniform Memory Access*(NUMA) e non SMP.

I processori AMD da tempo presentano questo tipo di architettura con le proprie interconnessioni *Hyper Transport* (HT), mentre Intel ha iniziato ad implementare NUMA nei propri *Quick Path Interconnect* (QPI). NUMA e SMP sono regolati in modo differente poichè è necessario considerare le *tipologie* del sistema durante l'assegnazione delle risorse per una applicazione.

THREAD

Nei sistemi operativi di Linux l'unità di esecuzione è conosciuta come *thread*. I thread hanno un contesto dei registri, una stack e un segmento del codice da eseguire su una CPU. È compito del sistema operativo (OS) programmare i thread sulle CPU disponibili.

Il sistema operativo massimizza l'uso della CPU attraverso un bilanciamento del carico dei thread su tutti i core disponibili. Poichè il compito principale del sistema operativo è quello di mantenere occupata la CPU, esso potrebbe non essere il più idoneo in relazione alla gestione delle prestazioni dell'applicazione. Spostando un thread dell'applicazione su una CPU di un altro socket, si potranno peggiorare le prestazioni più di quanto si possa verificare in relazione all'attesa di una CPU corrente, poichè le operazioni di accesso della memoria potrebbero rallentare drasticamente su tutti i socket. Per applicazioni ad elevate prestazioni è più idoneo determinare la posizione dei thread. Sezione 4.2, «Programmazione della CPU» contienene le informazioni necessarie su come assegnare le memoria e la CPU nel modo più opportuno per una esecuzione migliore dei thread dell'applicazione.

INTERRUPT

Uno degli eventi del sistema meno ovvi (ma importante) che possono interessare negativamente le prestazioni di una applicazione, è un *interrupt* (conosciuta in Linux come IRQ). Questi eventi sono gestiti dal sistema operativo e vengono usati dalle periferiche per segnalare l'arrivo di dati o il completamento di una operazione, come ad esempio le scritture della rete o un evento del timer.

La gestione di un interrupt da parte di un sistema operativo o una CPU che esegue un codice dell'applicazione, non interessa la funzionalità dell'applicazione, ma al contrario potrebbe riguardare le sue prestazioni. Questo capitolo affronta alcuni argomenti su come evitare un impatto negativo sulle prestazioni di una applicazione in presenza di interrupt.

4.1. TIPOLOGIA DELLA CPU

4.1.1. Tipologia NUMA e CPU

I primi processori erano di tipo *uniprocessor*, quindi il sistema era provvisto di una sola CPU. L'esecuzione di processi in parallelo era resa possibile tramite l'uso di un sistema operativo il quale smistava rapidamente la CPU da un thread di esecuzione (processo) ad un altro. Durante gli studi per migliorare le prestazioni, i designer notarono che aumentando la velocità dell'orologio per eseguire più velocemente le istruzioni, le prestazioni miglioravano ma solo fino ad un certo punto (generalmente limitazioni nella creazione di una frequenza dell'orologio stabile con la tecnologia corrente). Per migliorare le prestazioni è stata aggiunta una seconda CPU al sistema, permettendo così l'esecuzione parallela di due flussi. Questa tendenza di aggiungere processori è continuata con gli anni.

Numerosi sistemi con processori multipli venivano creati in modo che ogni CPU era in possesso dello stesso percorso logico per ogni posizione della memoria (generalmente un bus parallelo). In questo modo ogni CPU era in grado di accedere a qualsiasi posizione della memoria, usando la stessa quantità di tempo di qualsiasi altra CPU nel sistema. Questo tipo di architettura è conosciuta come sistema Symmetric Multi-Processor (SMP). SMP è ideale con un numero ristretto di CPU, ma superando un determinato numero (8 o 16), il numero di tracce parallele necessarie per un accesso uguale alla memoria utilizza una quantità di spazio troppo elevata, lasciando così meno spazio alla periferiche.

Due nuovi concetti uniti tra loro permettono di avere un numero più alto di CPU in un sistema:

1. Bus seriali

2. Tipologie NUMA

Un bus seriale è un percorso di comunicazione a linea singola con una velocità elevata dell'orologio in grado di trasferire dati in gruppi "o burst" di pacchetti con caratteristiche simili. In passato gli sviluppatori hardware hanno implementato i bus seriali come interconnessioni ad elevata velocità tra CPU, e tra i controllori della memoria e le CPU e altre periferiche. Questa impostazione permetteva l'uso di *una* traccia al posto di 32 e 34 tracce sulla scheda di *ogni* CPU per il sottosistema della memoria, riducendo così la quantità di spazio necessario sulla scheda.

Contemporaneamente gli sviluppatori hardware implementavano un numero sempre maggiore di transistor nello stesso spazio, riducendo così le dimensioni die. Al posto di inserire singole CPU direttamente sulla scheda principale, essi iniziarono ad inserirle in un pacchetto del processore utilizzandole così come processori multi-core. Successivamente al posto di fornire un accesso uniforme per la memoria da ogni pacchetto del processore, gli sviluppatori hanno iniziato ad implementare una strategia Non-Uniform Memory Access (NUMA), dove ogni combinazione pacchetto/socket presentava una o più aree specifiche della memoria per un accesso ad elevata velocità. Ogni socket possedeva altresì una interconnessione per altri socket per un accesso più lento alla memoria.

Per un esempio di implementazione NUMA semplice supponiamo di avere una scheda madre con due socket, dove ogni socket è stato popolato con un pacchetto quad-core. Ciò significa che il numero totale di CPU nel sistema è otto; quattro per ongi socket. Ogni socket presenta altresì un insieme di memoria assegnata con quattro gigabyte di RAM per un totale di otto gigabyte. Per questo esempio le CPU 0-3 sono posizionate nel socket 0, e le CPU 4-7 sono nel socket 1. Ogni socket in questo esempio corrisponde anche ad un nodo NUMA.

Per accedere alla memoria dall'insieme 0 potranno essere necessari tre cicli di orologio alla CPU 0: un ciclio per presentare l'indirizzo al controller della memoria, un ciclio per impostare l'accesso per la posizione della memoria e un ciclio per leggere o scrivere sulla posizione. Tuttavia potrebbe richiedere alla CPU 4 fino a sei cicli per accedere alla memoria dalla stessa posizione, poichè posizionata su un secondo socket, essa avrà la necessità di superare due controllori della memoria: il controllore della

memoria locale sul socket 1 e successivamente il controllore della memoria remota sul socket 0. Se la memoria viene contesa (e cioè se più di una CPU cerca di accedere alla stessa posizione contemporaneamente), i controllori avranno la necessità di regolare e serializzare l'accesso alla memoria, richiedendo così un tempo maggiore. Tale processo viene sommato alla consistenza della cache (assicurare che le cache delle CPU locali contangano dati uniformi per la stessa posizione della memoria) e complica maggiormente il processo.

Gli ultimissimi processori "high-end" di Intel (Xeon) e AMD (Opteron) presentano un NUMA. I processori AMD utilizzano una interconnessione conosciuta come HyperTransport o HT, mentre Intell utilizza una QuickPath Interconnect o QPI. Le interconnessioni differiscono dal tipo di collegamento fisico, memoria o dispositivi periferici, ma in effetti essi sono interruttori che permettono un accesso trasparente, da un dispositivo, ad un dispositivo connesso. In questo caso il termine trasparente si riferisce al fatto che non è presente alcuna API di programmazione speciale necessaria per l'utilizzo dell'interconnessione, non una opzione "senza alcun costo".

A causa della varietà di tipologie delle architetture di un sistema, non è pratico caratterizzare in modo specifico la penalità imposta da un accesso della memoria non-locale. Possiamo dire che ogni *segmento* "hop" in una interconnessione impone sempre una piccola penalità sulle prestazioni, per questo motivo riferire ad una posizione distante due interconnessioni dalla CPU corrente, impone un minimo di *2N + tempo del ciclo della memoria* per l'accesso, dove N è la penalità per segmento "hop".

Data la suddetta penalità, le applicazioni sensibili alle prestazioni dovrebbero evitare un accesso alla memoria remota in un sistema con una tipologia NUMA. Impostare l'applicazione in modo che essa si trovi in un nodo specifico, ed in grado di assegnare memoria dal nodo in questione.

Per fare questo è necessario essere a conoscenza di:

1. Cos'è la *tipologia* del sistema?
2. Dov'è l'applicazione attualmente eseguita?
3. Dov'è l'insieme di memoria più vicino?

4.1.2. Ottimizzazione delle prestazioni della CPU

Consultare questa sezione per capire come eseguire una ottimizzazione delle prestazioni della CPU e per una introduzione agli strumenti usati a tale scopo.

NUMA è stato originariamente usato per collegare un processore ad insiemi multipli di memoria. Con il miglioramento dei processi e la riduzione delle dimensioni dei circuiti integrati, è diventato possibile includere numerose CPU core in un pacchetto. Le suddette CPU core sono state clusterizzate in modo da avere un tempo d'accesso uguale ad un insieme della memoria locale, rendendo possibile così una condivisione della cache tra i core; tuttavia ogni 'segmento' presente in una interconnessione tra core, memoria e cache presenta una piccola penalità.

L'esempio di sistema presente in Figura 4.1, «Accesso alla memoria locale e remota con NUMA» riporta due nodi NUMA. Ogni nodi ha quattro CPU, un insieme di memoria ed un controller. Ogni CPU in un nodo ha un accesso diretto all'insieme della memoria su quel nodo. Seguendo le freccette sul Nodo 1 le fasi risultano essere:

1. Una CPU (qualsiasi tra 0-3) presenta un indirizzo per il controller della memoria locale.
2. Il controller della memoria imposta un accesso per l'indirizzo.
3. La CPU esegue operazioni di lettura o scrittura sull'indirizzo della memoria.

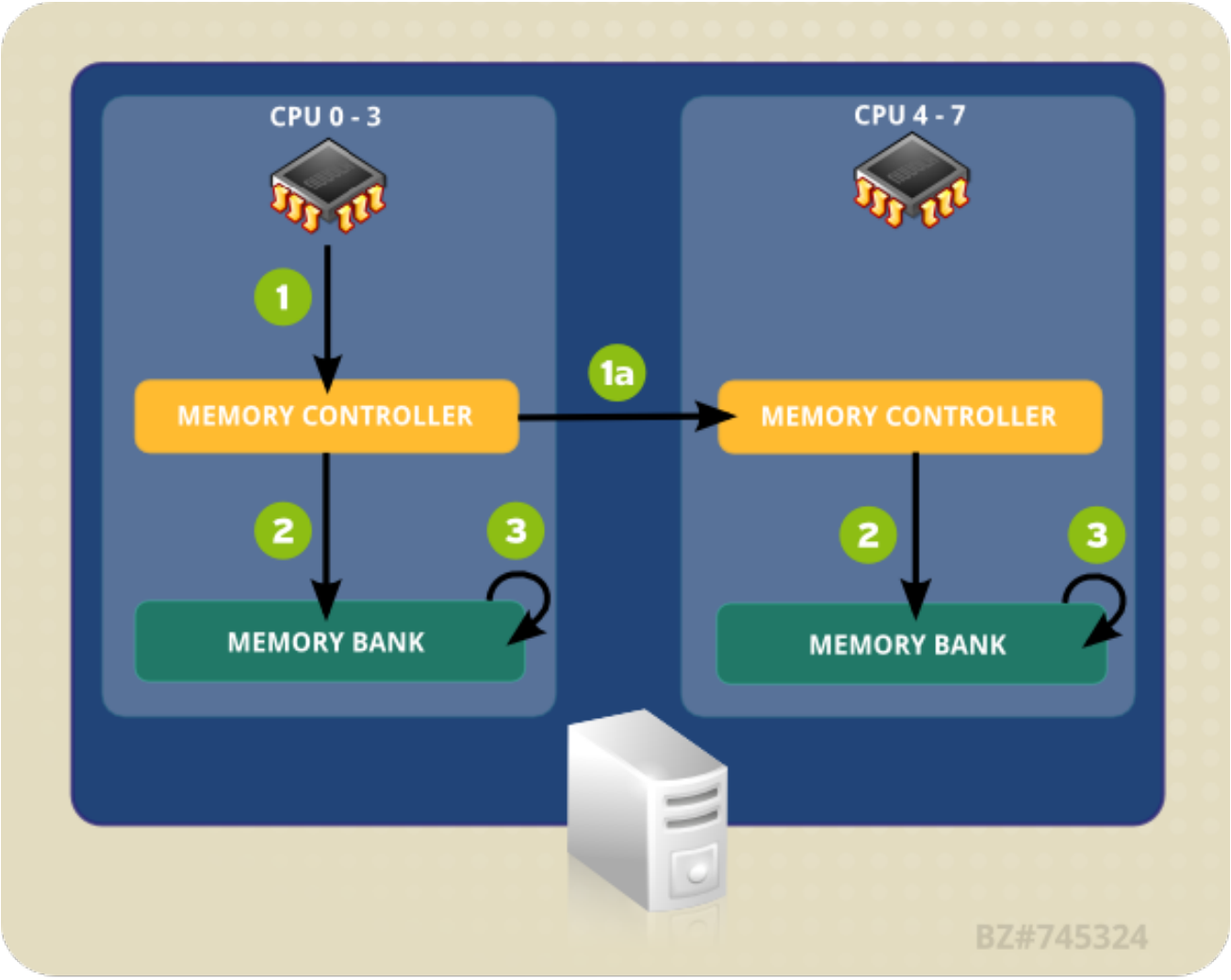


Figura 4.1. Accesso alla memoria locale e remota con NUMA

Tuttavia se una CPU di un nodo ha bisogno di accedere ad un codice all'interno di un insieme di memoria di un nodo NUMA diverso, il percorso da seguire sarà meno diretto:

1. Una CPU (qualsiasi tra 0-3) presenta un indirizzo della memoria remota per il controller della memoria locale.

    1. La richiesta di una CPU per l'indirizzo della memoria locale viene passato ad un controller della memoria remota, locale al nodo che contiene quell'indirizzo.

2. Il controller della memoria remota imposta un accesso per l'indirizzo della memoria remota.

3. La CPU esegue operazioni di lettura o scrittura sull'indirizzo della memoria remota.

Ogni azione deve passare attraverso numerosi controller di memoria, quindi l'accesso può richiedere il doppio del tempo se si cerca di accedere ad indirizzi di memorie remote. Quindi per mantenere un livello di prestazione desiderato in un sistema multi-core, assicurarsi che le informazioni siano passate il più efficientemente possibile usando il percorso più veloce o più corto.

Per configurare una applicazione per una prestazione della CPU ottimale è necessario sapere:

- la tipologia del sistema (collegamento dei componenti)
- il core sul quale viene eseguita l'applicazione, e

- la posizione dell'insieme di memoria più vicino.

Red Hat Enterprise Linux 6 rende disponibili alcuni strumenti per poter eseguire una ottimizzazione del sistema in base alle informazioni disponibili. Le seguenti sezioni forniscono una panoramica sugli strumenti utili per una ottimizzazione delle prestazioni della CPU.

4.1.2.1. Impostazione affinità della CPU con taskset

taskset ripristina ed imposta l'affinità della CPU di un processo in esecuzione (in base all'ID del processo). Può essere usato anche per lanciare un processo con una affinità della CPU conosciuta, associando il processo specifico ad una CPU o insieme di CPU. Tuttavia **taskset** non è in grado di garantire una assegnazione della memoria locale. Per benefici aggiuntivi dovuti all'assegnazione della memoria locale è consigliato usare **numactl** al posto di **taskset**; per maggiori informazioni consultare Sezione 4.1.2.2, «Controllo politica di NUMA con **numactl**».

L'affinità della CPU è rappresentata con un bitmask. Il bit con un ordine più basso corrisponde alla prima CPU logica, mentre il bit con l'ordine più alto corrisponde all'ultima CPU logica. Queste maschere vengono generalmente riportate in esadecimali, quindi **0x00000001** rappresenta il processore 0 e **0x00000003** rappresenta i processori 0 e 1.

Per impostare una affinità della CPU di un processo in esecuzione eseguire il seguente comando, sostituendo *mask* con la maschera del processore o processori con i quali desiderate associare il processo, e *pid* con l'ID del processo al quale desiderate modificare l'affinità.

```
# taskset -p mask pid
```

Per lanciare un processo con una data affinità eseguire il seguente comando, sostituendo *mask* con la maschera del processore o processori con i quali desiderate associare il processo, e *program* con il processo, le opzioni e gli argomenti del programma da eseguire.

```
# taskset mask -- program
```

Al posto di specificare i processori attraverso un bitmask sarà possibile specificare l'opzione **-c** per fornire un elenco delimitato da virgole di processori separati, o di una gamma di processori:

```
# taskset -c 0,5,7-9 -- myprogram
```

Maggiori informazioni su **taskset** sono disponibili attraverso la pagina man: **man taskset**.

4.1.2.2. Controllo politica di NUMA con numactl

numactl esegue i processi con una programmazione specifica o utilizzando una politica di posizionamento della memoria. La politica selezionata viene impostata per quel processo e per tutti i suoi processi figlio. **numactl** è in grado altresì di impostare una politica persistente per segmenti di memoria condivisi o file, e l'affinità della CPU e della memoria di un processo. Per determinare la tipologia del sistema esso utilizza il file system **/sys**.

Il file system **/sys** contiene informazioni utili sul collegamento delle CPU, memoria e dispositivi periferici con le interconnessioni NUMA. In modo specifico la directory **/sys/devices/system/cpu** contiene le informazioni sul collegamento delle CPU di un sistema. La directory **/sys/devices/system/node** contiene le informazioni sui nodi NUMA presenti nel sistema e sulle distanze che intercorrono tra i nodi.

In un sistema NUMA, maggiore è la distanza presente tra un processore ed un insieme di memoria e più lenta è la velocità d'accesso del processore per l'insieme in questione. A tale scopo configurare applicazioni sensibili alle prestazioni per poter assegnare memoria dall'insieme di memoria più vicino.

È consigliato anche configurare applicazioni "performance-sensitive" per eseguire un certo numero di core, in particolare nel caso di applicazioni multi-thread. Poichè le cache del primo livello sono generalmente piccole in presenza di thread multipli in esecuzione in un singolo core, ogni thread potrebbe espellere i dati memorizzati in cache usati da un thread precedente. Quando un sistema operativo cerca di eseguire un certo numero di compiti tra i thread, i quali a loro volta continuano ad espellere i dati usati dai thread precedenti, una grossa quantità di tempo di esecuzione verrà usata per la sostituzione della linea di cache. Questo problema si chiama *cache thrashing*. Per questo motivo è consigliato associare una applicazione multi-thread ad un nodo e non ad un singolo core, poichè così facendo permetterete ai thread di condividere le linee in cache su livelli multipli (prima, seconda e cache dell'ultimo livello) e minimizzerete la necessità di operazioni per il riempimento della cache. Tuttavia l'associazione di una applicazione ad un core singolo può garantire un buon livello di prestazioni se tutti i thread accedono agli stessi dati presenti in cache.

numactl permette di associare una applicazione ad un nodo NUMA o core particolare, e di assegnare la memoria associata con un core o insieme di core all'applicazione in questione. Alcune opzioni utili disponibili con numactl sono:

--show

Mostra le impostazioni della politica NUMA del processo corrente. Questo parametro non ha bisogno di parametri aggiuntivi e può essere usato nel modo seguente: **`numactl --show`**.

--hardware

Mostra un inventario dei nodi disponibili sul sistema.

--membind

Assegna solo la memoria dai nodi specificati. Se in uso, il processo di assegnazione fallirà se la memoria sui nodi in questione non è sufficiente. Il formato per questo parametro è **`numactl --membind=`*`nodes program`***, dove *nodes* è l'elenco dei nodi usati per assegnare la memoria e *program* è il programma usato per i requisiti di memoria che i nodi dovranno assegnare. È possibile indicare i numeri dei nodi con un elenco separato da virgole, una gamma o una combinazione dei due. Maggiori informazioni sono disponibili sulle pagine man di **numactl**: **`man numactl`**.

--cpunodebind

Esegue solo un comando (e i processi figli relativi) sulle CPU che appartengono ai nodi specificati. Il formato di questo parametro è il seguente **`numactl --cpunodebind=`*`nodes program`***, dove *nodes* è l'elenco dei nodi specificati usati per indicare le CPU per l'assegnazione dei programmi desiderati (*program*). È possibile indicare i numeri dei nodi con un elenco separato da virgole, una gamma o una combinazione dei due. Maggiori informazioni sono disponibili sulle pagine man di **numactl**: **`man numactl`**.

--physcpubind

Esegue solo un comando (e i processi figli relativi) sulle CPU specificate. Il formato di questo parametro è il seguente **`numactl --physcpubind=`*`cpu program`***, dove *cpu* è l'elenco delimitato da virgole dei numeri delle CPU fisiche, come riportato nei campi dei processori di **`/proc/cpuinfo`**, e *program* è il programma da eseguire solo sulle CPU in questione. È possibile specificare le CPU in relazione al **`cpuset`** corrente. Consultare le pagine man di **numactl** per maggiori informazioni: **`man numactl`**.

--localalloc

Specifica che la memoria deve sempre essere assegnata sul nodo corrente.

--preferred

Quando possibile assegna la memoria sul nodo specificato, In caso contrario usare altri nodi. Questa opzione accetta solo il numero di un singolo nodo, come ad esempio: **`numactl --preferred=`*node***. Consultare la pagina man di numactl per maggiori informazioni: **`man numactl`**.

La libreria **libnuma** inclusa nel pacchetto numactl offre una interfaccia di programmazione semplice per la politica NUMA supportata dal kernel. Utile per una regolazione più dettagliata rispetto all'utilità **numactl**. Per maggiori informazioni consultare la pagina man relativa: **`man numa(3)`**.

4.1.3. numastat

IMPORTANTE

In precedenza **numastat** era uno script Perl scritto da Andi Kleen, mentre ora è stato riscritto per Red Hat Enterprise Linux 6.4.

Anche se il comando predefinito (**`numastat`** senza alcuna opzione o parametri) mantiene una compatibilità con la versione precedente, l'uso di opzioni o parametri con questo comando modificherà sensibilmente il formato e il contenuto dell'output.

numastat mostra le informazioni relative alla memoria (ad esempio i successi ed i fallimenti della memoria) per processi e sistemi operativi in base al per-NUMA-nodo. Per impostazione predefinita l'esecuzione di **`numastat`** mostra il numero di pagine di memoria occupate dalle seguenti categorie di eventi per ogni nodo.

Prestazioni ottimali della CPU vengono indicate da valori **`numa_miss`** e **`numa_foreign`** bassi.

Questa versione aggiornata di **numastat** indica anche se la memoria del processo viene condivisa dall'intero sistema o se risulta centralizzata su nodi specifici che utilizzano **numactl**.

Eseguire un riferimento incrociato dell'output **numastat** con l'output **top** per-CPU, per verificare che i thread del processo siano in esecuzione sugli stessi nodi ai quali è stata assegnata la memoria.

Categorie di verifica predefinite

numa_hit

Numero di tentativi di assegnazione riusciti per il nodo.

numa_miss

Numero di tentativi di assegnazione ad un altro nodo allocati a questo nodo a causa di una quantità di memoria limitata sul nodo desiderato. Ogni evento **`numa_miss`** ha un evento **`numa_foreign`** corrispondente su un altro nodo.

numa_foreign

Numero di assegnazioni intese inizialmente per questo nodo allocate invece ad un altro nodo. Ogni evento **`numa_foreign`** ha un evento **`numa_miss`** su un altro nodo.

interleave_hit

Numero di tentativi di assegnazione della politica interleave per questo nodo che hanno avuto successo.

local_node

Numero di volte che un processo sul nodo ha assegnato con successo la memoria a questo nodo.

other_node

Numero di volte che un processo su un altro nodo ha assegnato la memoria a questo nodo.

L'uso delle seguenti opzioni modifica le unità visualizzate in megabyte di memoria (approssimate a due cifre decimali), e modifica altri comportamenti **numastat** specifici come di seguito riportato.

-c

Riduce orizzontalmente la tabella delle informazioni. Utile su sistemi con un numero molto grande di nodi NUMA, ma lo spazio tra colonne e la loro larghezza può essere imprevedibile. Usando questa opzione la quantità di memoria viene approssimata al megabyte per vicino.

-m

Mostra le informazioni sull'uso della memoria dell'intero sistema in base al nodo. Simile alle informazioni disponibili in **/proc/meminfo**.

-n

Mostra le stesse informazioni del comando originale **numastat** (numa_hit, numa_miss, numa_foreign, interleave_hit, local_node, and other_node), con un formato aggiornato, usando i megabyte come unità di misura.

-p ***pattern***

Mostra le informazioni della memoria per-node per il pattern specificato. Se il valore di *pattern* è composto da cifre, **numastat** assume che si tratti di un identificatore di un processo numerico. In caso contrario **numastat** ricerca le righe del comando del processo per il pattern specificato.

Gli argomenti della linea di comando inseriti dopo il valore dell'opzione **-p** vengono considerati pattern aggiuntivi da filtrare. Pattern aggiuntivi aumentano e non diminuiscono i criteri di filtraggio.

-s

Riporta i dati visualizzati in ordine decrescente in modo da riportare prima le utenze più grandi della memoria (in base alla colonna **total**).

Facoltativamente specificare *node*, così facendo la tabella verrà ordinata in base ai valori della colonna *node*. Con questa opzione il valore di *node* deve seguire immediatamente l'opzione **-s** come mostrato qui di seguito:

```
numastat -s2
```

Non includere alcuno spazio tra questa opzione ed il proprio valore.

-v

Mostra informazioni più verbose. Le informazioni per processi multipli mostreranno informazioni dettagliate per ogni processo.

-v

Mostra le informazioni sulla versione di **numastat**.

-z

Omette dalle informazioni visualizzate le colonne e le righe della tabella con un valore zero. Da notare che valori vicini allo zero approssimati a zero per facilitare la visualizzazione, non saranno omessi dall'output.

4.1.4. NUMA Affinity Management Daemon (numad)

numad è un demone di gestione dell'affinità NUMA automatico. Esso controlla la tipologia NUMA e l'uso delle risorse all'interno del sistema, in modo da migliorare dinamicamente la gestione e l'assegnazione delle risorse NUMA (e quindi delle prestazioni del sistema).

In base al carico di lavoro del sistema **numad** può fornire miglioramenti delle prestazioni fino ad un massimo del 50%. Per raggiungere questi livelli **numad** accede periodicamente alle informazioni disponibili sul file system `/proc`, per controllare la disponibilità delle risorse in base al nodo. Successivamente il demone cercherà di posizionare un numero consistente di processi sui nodi NUMA con una memoria sufficientemente allineata e in possesso di risorse della CPU, in modo da avere una prestazione ottimale. Attualmente i limiti per la gestione dei processi sono di un minimo del 50% di una CPU e di almeno 300 MB di memoria. **numad** cerca di mantenere un livello di utilizzo delle risorse, bilanciando le assegnazioni spostando i processi tra i nodi NUMA.

numad fornisce anche un servizio di pre-assegnazione consultabile dai vari sistemi di gestione dei compiti, che fornisce assistenza con l'associazione iniziale delle risorse della memoria e della CPU ai rispettivi processi. Questo servizio è disponibile anche se **numad** non è in esecuzione come demone sul sistema. Consultare la pagina man per maggiori informazioni su come usare l'opzione `-w` per una assistenza pre-assegnazione: `man numad`.

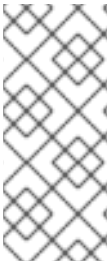
4.1.4.1. Benefici di numad

numad apporta miglioramenti principalmente su sistemi con processi in esecuzione per periodi estesi che utilizzano una quantità molto elevata di risorse, ed in particolare quando i suddetti processi sono contenuti in un sottoinsieme di risorse totali del sistema.

numad puo essere anche idoneo per applicazioni che utilizzano le risorse di nodi multipli NUMA. Tuttavia i benefici forniti da **numad** diminuiscono all'aumentare del numero di risorse consumate in un sistema.

La possibilità che **numad** migliori le prestazioni quando i processi vengono eseguiti solo per pochi minuti, o in presenza di un basso consumo di risorse, è molto bassa. Altresì, sistemi con pattern di accesso alla memoria non prevedibili potrebbero non trarre alcun beneficio dall'uso di **numad**.

4.1.4.2. Modalità di operazione



NOTA

Le statistiche sul conteggio della memoria del kernel potrebbero confondere **numad** dopo numerose operazioni di merge di gran parte della memoria da parte del demone KSM. Il demone KSM sarà in grado di riconoscere NUMA nelle release future. Tuttavia attualmente se il sistema presenta una quantità molto grande di memoria, sarà possibile migliorare le prestazioni disabilitando il demone KSM.

numad può essere utilizzato in due modi:

- come un servizio
- come un eseguibile

4.1.4.2.1. Usare numad come servizio

Durante l'esecuzione di **numad** esso cercherà di eseguire una regolazione dinamica del sistema in base al carico di lavoro del sistema.

Per avviare il servizio eseguire:

```
# service numad start
```

Per rendere il servizio persistente dopo riavvii multipli eseguire:

```
# chkconfig numad on
```

4.1.4.2.2. Usare numad come eseguibile

Per usare **numad** come un eseguibile eseguire::

```
# numad
```

numad continuerà la sua esecuzione fino a quando non verrà arrestato. Durante l'esecuzione le sue attività verranno registrate su **/var/log/numad.log**.

Per limitare la gestione di **numad** ad un processo specifico avviatelo usando le seguenti opzioni.

```
# numad -S 0 -p pid
```

-p pid

Aggiunge il *pid* specifico ad un elenco di inclusione esplicito. Il processo specificato non verrà gestito fino a quando non soddisfa il limite significativo del processo **numad**.

-S mode

Il parametro ***-S*** specifica il tipo di scansione. Impostando **0**, la gestione di **numad** verrà limitata a processi esplicitamente inclusi.

Per arrestare **numad** eseguite:

```
# numad -i 0
```

L'arresto di **numad** non rimuoverà le modifiche eseguite per migliorare l'affinità di NUMA. Se l'utilizzo del sistema varia significativamente, una nuova esecuzione di **numad** modificherà l'affinità migliorando così le prestazioni.

Per maggiori informazioni sulla disponibilità delle opzioni di **numad** consultare la pagina man **numad**: **`man numad`**.

4.2. PROGRAMMAZIONE DELLA CPU

Lo *scheduler* è responsabile per mantenere occupate le CPU del sistema. Lo scheduler di Linux implementa un numero di *politiche di programmazione*per mezzo delle quali è possibile determinare quando e per quanto tempo il thread viene eseguito su una CPU core particolare.

Le politiche di programmazione sono suddivise in due categorie principali:

1. Politiche realtime
    - SCHED_FIFO
    - SCHED_RR
2. Politiche normali
    - SCHED_OTHER
    - SCHED_BATCH
    - SCHED_IDLE

4.2.1. Politiche per una programmazione di tipo realtime

I thread realtime vengono programmati prima, mentre i thread normali sono programmati dopo la programmazione di tutti i thread realtime.

Le politiche *realtime* vengono implementate per compiti "time-critical" da completarsi senza alcuna interruzione.

`SCHED_FIFO`

Questa politica viene anche chiamata *static priority scheduling* poichè essa definisce una priorità fissa (tra 1 e 99) per ogni thread. Lo scheduler esegue la scansione di un elenco di thread SCHED_FIFO seguendo un ordine di priorità, e programma il thread con la priorità più alta pronto ad essere eseguito. Il thread verrà eseguito, arrestato, abbandonato o preceduto da un altro thread con priorità più alta.

Anche un thread realtime con priorità più bassa verrà programmato prima di qualsiasi altro thread con una politica diversa da realtime; se esiste solo un thread realtime, il valore della priorità **`SCHED_FIFO`** non avrà alcuna importanza.

`SCHED_RR`

Variante round-robin della politica **`SCHED_FIFO`**. I thread di **`SCHED_FIFO`** avranno una priorità fissa tra 1 e 99. Tuttavia i thread con la stessa priorità verranno programmati usando un criterio round-robin all'interno di un determinato quantum o periodo di tempo. La chiamata del sistema **`sched_rr_get_interval(2)`** ritorna un valore relativo al periodo di tempo, ma la durata di questo periodo non potrà essere impostata dall'utente. Questa politica è utile se desiderate eseguire thread multipli con la stessa priorità.

Per maggiori informazioni sulle semantiche definite delle politiche di programmazione relatime consultate lo *IEEE 1003.1 POSIX standard* delle interfacce del sistema — Realtime, disponibile su http://pubs.opengroup.org/onlinepubs/009695399/functions/xsh_chap02_08.html.

Per definire una priorità dei thread è consigliato usare una priorità bassa e successivamente aumentarla in presenza di una latenza. I thread relatime non hanno alcun limite di tempo come quelli normali; Thread **SCHED_FIFO** possono essere eseguiti fino a quando gli stessi non verranno bloccati, abbandonati o preceduti da altri thread con priorità più alte. Per questo motivo non è consigliato impostare la priorità su 99, poichè tale priorità posizionerà il processo allo stesso livello dei thread di migrazione e watchdog. Se i thread vengono bloccati poichè essi entrano in un loop, gli stessi non potranno essere eseguiti. Sistemi con un solo processore avranno una elevata possibilità di incontrare questo scenario.

Nel kernel di Linux la politica **SCHED_FIFO** include un meccanismo di limitazione della larghezza di banda. Tale meccanismo protegge i programmatori in modo da evitare un monopolio della CPU da parte di eventi realtime. Questo meccanismo può essere modificato attraverso i seguenti parametri del file system **/proc**:

/proc/sys/kernel/sched_rt_period_us

Definisce il periodo di tempo da considerare cento per cento larghezza di banda della CPU, in millisecondi ('us' l'equivalente più vicino a 'µs' in testo semplice). Il valore predefinito è 1000000µs, o 1 secondo.

/proc/sys/kernel/sched_rt_runtime_us

Definisce il periodo di tempo da conferire per l'esecuzione dei thread realtime in millisecondi ('us' l'equivalente più vicino a 'µs' in testo semplice). Il valore predefinito è 950000µs, o 0,95 secondi.

4.2.2. Politiche di programmazione normale

Sono disponibili tre politiche di programmazione normale: **SCHED_OTHER**, **SCHED_BATCH** e **SCHED_IDLE**. Tuttavia le politiche **SCHED_BATCH** e **SCHED_IDLE** sono relative a compiti con una priorità bassa e per questo motivo hanno poca rilevanza nella guida di ottimizzazione delle prestazioni.

SCHED_OTHER, o SCHED_NORMAL

La politica di programmazione predefinita. Questa politica usa un Completely Fair Scheduler (CFS) per fornire periodi di accesso imparziali per tutti i thread che utilizzano questa politica. CFS stabilisce un elenco di priorità dinamica in parte basato sul valore ***niceness*** di ogni thread del processo. (Consultare la *Deployment Guide* per maggiori informazioni su questo parametro e sul file system **/proc**.) Ciò conferisce agli utenti un livello indiretto di controllo sulla priorità dei processi, ma l'elenco di priorità dinamico può essere modificato direttamente solo dal CFS.

4.2.3. Selezione politica

La selezione di una politica dello scheduler per i thread di una applicazione non è sempre un compito semplice. In generale, le politiche realtime devono essere usate per compiti importanti o soggetti ad una scadenza da programmare velocemente, e non eseguibili per periodi estesi di tempo. Politiche normali generalmente hanno risultati di output netto dei dati migliori rispetto alle politiche realtime, poichè esse lasciano allo scheduler il compito di eseguire i thread in modo più efficiente (e cioè non hanno alcuna necessità di assegnare la precedenza ai processi).

Se gestite un numero molto grande di thread e l'output netto dei dati è il parametro a voi prioritario (pacchetti di rete per secondo, scritture sul disco ecc.), allora è consigliato usare **SCHED_OTHER** e lasciare al sistema la gestione dell'uso della CPU.

Se invece il tempo di risposta (latenza) è il parametro prioritario, usate **SCHED_FIFO**. Se siete in possesso di un numero limitato di thread, isolate un socket della CPU e spostate i thread sui core del socket. Così facendo nessun thread entrerà in competizione sui core.

4.3. INTERRUPT E OTTIMIZZAZIONE IRQ

Un interrupt request (IRQ) è una richiesta per il servizio inviata a livello hardware. Gli interrupt possono essere inviati da un hardware specifico o attraverso un bus hardware come pacchetto di informazioni (un Message Signaled Interrupt, o MSI).

In presenza di interrupt, alla ricezione di un IRQ si verificherà uno smistamento ad un stato interrupt. Il codice di smistamento degli interrupt del kernel recupera il valore IRQ insieme all'elenco associato di Interrupt Service Routines (ISR) registrati, invocando a turno ogni ISR. ISR riconoscerà il segnale e ignorerà gli interrupt ridondanti dello stesso IRQ, mettendo in coda un gestore per la processazione dell'interrupt impedendo ad ISR di ignorare gli interrupt futuri.

Il file **/proc/interrupts** elenca il numero di interrupt per CPU per dispositivo I/O. Mostra il numero di IRQ, il numero dell'interrupt gestito da ogni CPU core, il tipo di interrupt ed un elenco delimitato da virgole di driver registrati per ricevere il segnale. (Consultare la pagina man proc(5) per maggiori informazioni: **man 5 proc**)

Gli IRQ presentano una proprietà "affinity" associata, ***smp_affinity***, la quale definisce i CPU core in grado di eseguire ISR per l'IRQ in questione. Usare questa proprietà per migliorare le prestazioni dell'applicazione assegnando sia l'affinità dell'interruzione che quella del thread dell'applicazione ad uno più CPU core. Ciò permette di avere una linea di condivisione della cache tra l'interrupt specificato ed i thread dell'applicazione.

Il valore dell'affinità dell'interrupt per un particolare numero IRQ è archiviato nel file **/proc/irq/*IRQ_NUMBER*/smp_affinity** associato, disponibile e modificabile dall'utente root. Il valore archiviato in questo file è una maschera di bit esadecimale che rappresenta tutti i CPU core presenti nel sistema.

Per esempio, per impostare l'affinità dell'interrupt per il driver Ethernet su un server con quattro CPU core, determinare prima il numero IRQ associato con il driver Ethernet:

```
# grep eth0 /proc/interrupts
32:   0     140      45       850264      PCI-MSI-edge      eth0
```

Utilizzare il valore IRQ per rilevare il file ***smp_affinity*** appropriato:

```
# cat /proc/irq/32/smp_affinity
f
```

Il valore predefinito per smp_affinity è **f**, ciò significa che IRQ può essere servito su qualsiasi CPU del sistema. L'impostazione di questo valore su **1** indicherà che solo la CPU 0 può servire questo interrupt:

```
# echo 1 >/proc/irq/32/smp_affinity
# cat /proc/irq/32/smp_affinity
1
```

Usare le virgole per delimitare i valori di ***smp_affinity*** per gruppi a 32-bit discreti. Questa impostazione è necessaria per sistemi con un numero di core maggiore di 32. Di seguito viene riportato un esempio nel quale IRQ 40 viene servito su tutti i core di un sistema a 64-core:

```
# cat /proc/irq/40/smp_affinity
ffffffff,ffffffff
```

Per servire IRQ 40 solo nella parte alta di un 32-core in un sistema a 64-core, eseguire quanto di seguito riportato:

```
# echo 0xffffffff,00000000 > /proc/irq/40/smp_affinity
# cat /proc/irq/40/smp_affinity
ffffffff,00000000
```

NOTA

Su sistemi che supportano gli *interrupt steering*, se modificate ***smp_affinity*** di un IRQ, la decisione di servire un interrupt con una CPU particolare viene presa a livello hardware senza alcun intervento da parte del kernel.

4.4. MIGLIORAMENTI NUMA IN RED HAT ENTERPRISE LINUX 6

Red Hat Enterprise Linux 6 introduce un certo numero di miglioramenti per trarre il maggior beneficio dalle potenzialità di un hardware altamente scalabile. Questa sezione fornisce una panoramica dettagliata dei miglioramenti più importanti delle prestazioni relative a NUMA, presenti con Red Hat Enterprise Linux 6.

4.4.1. Ottimizzazione bare-metal e scalabilità

4.4.1.1. Miglioramenti sul riconoscimento della tipologia

I miglioramenti di seguito riportati permettono a Red Hat Enterprise Linux di rilevare informazioni dettagliate relative all'architettura e all'hardware, migliorando la possibilità di ottimizzare automaticamente la processazione del sistema.

rilevamento della tipologia migliorato

Ciò permette ad un sistema operativo di rilevare informazioni dettagliate sull'hardware (come ad esempio CPU logiche, hyper thread, core, socket, nodi NUMA e tempi di accesso tra i nodi) al momento dell'avvio e di ottimizzare la processazione sul sistema.

completely fair scheduler

Questa nuova modalità d'accesso assicura una condivisione uniforme del tempo di esecuzione tra i processi interessati. Unendo questa impostazione con il rilevamento della tipologia sarà possibile programmare i processi su CPU all'interno dello stesso socket, evitando così la necessità di avere un accesso della memoria remota, e assicurando che il contenuto presente in cache possa essere conservato quando possibile.

`malloc`

`malloc` è stato ora ottimizzato ed è in grado di assicurare che le regioni della memoria assegnate ad un processo, siano il più vicine possibili al core sul quale il processo risulta essere in esecuzione. Questa impostazione migliora la velocità di accesso della memoria.

Assegnazione buffer I/O skbuff

In modo simile a `malloc`, è stata eseguita una sua ottimizzazione per utilizzare la memoria fisicamente vicina alle operazioni I/O di gestione della CPU, come ad esempio gli interrupt del dispositivo.

affinità interrupt del dispositivo

È possibile utilizzare le informazioni registrate dai driver dei dispositivi relative alla gestione degli interrupt da parte delle CPU. Queste informazioni possono essere usate per limitare la gestione degli interrupt alle CPU presenti sullo stesso socket fisico, conservando così l'affinità della cache e limitando il volume delle comunicazioni tra i socket.

4.4.1.2. Miglioramento nella sincronizzazione tra processori multipli

L'organizzazione dei compiti tra i processori multipli richiede operazioni frequenti che utilizzano un tempo molto elevato per assicurare l'esecuzione dei processi in parallelo senza compromettere l'integrità dei dati. Red Hat Enterprise Linux include i seguenti miglioramenti per le aree di seguito riportate:

Blocchi Read-Copy-Update (RCU)

Generalmente il 90% di blocchi vengono usati per processi di sola lettura. l'RCU locking rimuove la necessità di ottenere un blocco per un accesso esclusivo quando i dati usati non sono stati modificati. Questa modalità viene ora implementata nell'assegnazione della memoria cache della pagina: il blocco viene usato ora solo per operazione di allocazione/deallocazione.

algoritmi per-CPU e per-socket

Numerosi algoritmi sono stati aggiornati per poter eseguire un blocco coordinato tra CPU in coperazione tra loro presenti sullo stesso socket. Questa operazione permette di avere un blocco più dettagliato. Numerosi spinlock sono stati rimossi e sostituiti da metodi di blocco per-socket. L'aggiornamento delle zone dell'allocatore di memoria e degli elenchi delle pagine relativi, permette di usare una logica di assegnazione con un sottoinsieme più efficiente di strutture per la mappatura dei dati della memoria durante operazioni di allocazione/deallocazione.

4.4.2. Miglioramenti virtualizzazione

Poichè KVM utilizza le funzionalità del kernel, i guest virtualizzati basati su KVM possono trarre beneficio da tutte le ottimizzazioni bare-metal. Red Hat Enterprise Linux include anche un certo numero di miglioramenti permettendo così ai guest virtualizzati di raggiungere un livello di prestazione pari ad un sistema bare-metal. Questi miglioramenti sono rivolti al percorso I/O nell'accesso di rete e storage, e permette ai carichi molto elevati, come quelli del database e file-serving, di usufruire dell'implementazione virtualizzata. I miglioramenti specifici a NUMA in grado di migliorare le prestazioni di sistemi virtualizzati includono:

CPU pinning

È possibile associare l'esecuzione dei guest virtuali ad un socket specifico per ottimizzare la cache locale in uso, rimuovendo così la necessità di utilizzare comunicazioni inter-socket costose e l'accesso alla memoria remoto.

transparent hugepages (THP)

Con THP abilitato, il sistema esegue automaticamente richieste di allocazione della memoria NUMA-riconosciuta per quantità di memoria adiacente molto grande, riducendo così la contesa del blocco e le operazioni di gestione della memoria translation lookaside buffer (TLB) necessarie,

aumentando le prestazioni fino al 20% nei guest virtuali.

Implementazione I/O basata sul kernel

Il sottosistema I/O del guest virtuale è ora implementato nel kernel, ed è in grado di ridurre l'accesso alla memoria e la comunicazione inter-node, impedendo l'esecuzione di una quantità molto grande di scambio del contesto, ed un sovraccarico delle comunicazioni e sincronizzazione.

CAPITOLO 5. MEMORIA

Consultare questo capitolo per una panoramica sulle funzioni di gestione della memoria con Red Hat Enterprise Linux e su come utilizzare le suddette funzioni per ottimizzarne l'uso.

5.1. HUGE TRANSLATION LOOKASIDE BUFFER (HUGETLB)

Gli indirizzi della memoria fisica sono tradotti in indirizzi della memoria virtuale come parte del processo di gestione della memoria. La relazione mappata di indirizzi fisici e virtuali viene archiviata in una struttura dati conosciuta come tabella della pagina. Poichè la lettura della tabella per ogni mappatura è un processo che richiede molto tempo e molte risorse, è disponibile una cache per indirizzi usati più di recente. Questa cache viene chiamata Translation Lookaside Buffer (TLB).

Tuttavia il TLB può solo memorizzare in cache un certo numero di mappature degli indirizzi. Se una mappatura desiderata non è pesente nel TLB, sarà necessario leggere la tabella della pagina per determinare la mappatura dell'indirizzo fisico a virtuale. Questo processo è conosciuto come "TLB miss". Le applicazioni con requisiti di memoria molto grandi possono essere interessate maggiormente da un TLB miss, rispetto alle applicazioni con requisiti di memoria minori a causa della relazione presente tra i requisiti di memoria e la dimensione delle pagine usate per memorizzare in cache le mappature degli indirizzi nel TLB. Poichè ogni TLB miss richiede un processo di lettura della tabella, è importante evitare questi processi quando possibile.

Huge Translation Lookaside Buffer (HugeTLB) permette una gestione della memoria con segmenti molto grandi per l'archiviazione contemporanea in cache di un numero maggiore di mappature dell'indirizzo. Questa impostazione riduce la possibilita di risultati negativi TLB, e permette di avere migliori prestazioni con applicazioni con requisiti di memoria molto grandi.

Le informazioni sulla configurazione di HugeTLB sono disponibili nella documentazione del kernel: **`/usr/share/doc/kernel-doc-`*`version`*`/Documentation/vm/hugetlbpage.txt`**

5.2. HUGE PAGES E TRANSPARENT HUGE PAGES

La memoria viene gestita in blocchi conosciuti come *pagine*. Una pagina è composta da 4096 byte. 1MB di memoria equivale a 256 pagine; 1GB di memoria equivale a 256,000 pagine, ecc. Le CPU presentano una *unità di gestione della memoria*interna che contiene un elenco di pagine, con ogni pagina riferita come *voce tabella della pagina*

Sono disponibili due metodi di gestione di memoria molto grande del sistema:

- Aumento del numero di voci della tabella della pagina nell'unità di gestione della memoria hardware
- Aumento della dimensione della pagina

Il primo metodo è molto costoso poichè l'unità di gestione della memoria hardware nei processori moderni supporta solo centinaia o migliaia di voci. Altresì, gli algoritmi di gestione della memoria e hardware che operano correttamente con migliaia di pagine (megabyte di memoria), possono non operare in modo ottimale con milioni (o miliardi) di pagine. Tale impostazione può presentare alcuni problemi: quando una applicazione ha bisogno di usare un numero di pagine di memoria maggiore rispetto al numero supportato dall'unità di gestione, il sistema implementa una gestione della memoria basata sul software, causando una esecuzione più lenta dell'intero sistema.

Red Hat Enterprise Linux 6 implementa il secondo metodo tramite l'uso delle *huge pages*.

Le huge page sono blocchi di memoria con una misura di 2MB e 1GB. Le tabelle usate dalle pagine di 2MB sono idonee per la gestione di gigabyte multipli di memoria, mentre quelle usate per pagine di 1GB sono idonee per misure pari a terabyte di memoria.

Assegnare le huge page al momento dell'avvio. La gestione manuale delle huge page è difficoltosa poichè spesso sono necessarie modifiche importanti al codice per permettere un loro uso effettivo. Per questo Red Hat Enterprise Linux 6 permette anche l'uso delle *transparent huge pages* (THP). THP è un livello di astrazione in grado di automatizzare numerosi aspetti relativi alla creazione, gestione ed utilizzo delle huge page.

THP minmizza la maggior parte della complessità nell'uso delle huge page da parte degli amministratori del sistema o sviluppatori. L'obiettivo di THP è quello di migliorare le prestazioni; gli sviluppatori (sia quelli della comunità che quelli di Red Hat) hanno eseguito test e ottimizzato THP su una gamma estesa di sistemi, configurazioni, applicazioni, e carichi di lavoro. Ciò permette alle impostazioni predefinite di THP di migliorare le prestazioni di numerose configurazioni.

Da notare che THP attualmente mappa solo regioni della memoria anonime come ad esempio lo spazio di stack e heap.

5.3. COME USARE VALGRIND PER UNA ANALISI SULL'USO DELLA MEMORIA

Valgrind è un framework in grado di fornire una strumentazione ai binari dello spazio utente. È presente con un numero di strumenti utilizzabili per il tracciamento e l'analisi delle prestazioni dei programmi. Gli strumenti presenti in questa sezione forniscono una analisi in grado di assistere l'utente al rilevamento degli errori della memoria, come ad esempio l'uso di memoria non inizializzata e una assegnazione incorretta o una deallocazione della memoria stessa. Il tutto è presente con il pacchetto valgrind e può essere eseguito con il seguente comando:

```
valgrind --tool=toolname program
```

Sostituire *toolname* con il nome dello strumento desiderato (per una descrizione della memoria, **memcheck**, **massif**, o **cachegrind**), e *program* con il programma desiderato da usare con Valgrind. Attenzione, l'uso di Valgrind causerà una esecuzione più lenta del programma rispetto alle condizioni normali.

Una panoramica sulle capacità di Valgrind è disponibile in Sezione 3.5.3, «Valgrind». Per maggiori informazioni sui plugin disponibili per Eclipse consultate la *Developer Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/. Per visualizzare la documentazione disponibile utilizzare il comando **man valgrind**; questo comando è disponibile se avete installato il pacchetto valgrind o presente nelle seguenti posizioni:

- **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**, and
- **/usr/share/doc/valgrind-*version*/html/index.html**.

5.3.1. Come usare Memcheck per una analisi sull'uso della memoria

Memcheck è lo strumento di Valgrind predefinito e può essere eseguito con **valgrind *program***, senza specificare **--tool=memcheck**. Esso rileva e riporta un certo numero di errori della memoria difficili da rilevare e diagnosticare, come ad esempio un accesso della memoria non previsto, l'uso di valori non inizializzati o non definiti, memoria heap resa disponibile non correttamente, puntatori sovrapposti e perdite di memoria. Se utilizzate Memcheck i programmi verranno eseguiti da dieci a trenta volte più lentamente rispetto ad una loro esecuzione normale.

Memcheck ritorna errori specifici in base al tipo di problema rilevato. Questi errori vengono riportati in dettaglio nella documentazione di Valgrind inclusa in **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**.

Da notare che Memcheck riporta solo questi errori — e non impedisce la loro generazione. Se un programma esegue un accesso alla memoria tale da verificarsi un errore di segmentazione, questo errore si verificherà anche se utilizzate Memcheck. Detto questo, Memcheck registrerà un messaggio immediatamente prima alla presenza dell'errore.

Memcheck fornisce le opzioni della linea di comando necessarie per il controllo del processo. Alcune di queste opzioni disponibili sono:

--leak-check

Se abilitato Memcheck va alla ricerca di perdite di memoria quando il programma client termina la sua esecuzione. Il valore predefinito è **summary**, e il suo output riporta il numero di perdite trovate. Altri valori possibili sono **yes** e **full**, entrambi riportano le informazioni su ogni perdita individuale e **no** disabilita questo tipo di controllo.

--undef-value-errors

Se abilitato (impostato su **yes**), Memcheck riporta gli errori se vengono usati valori non identificati. Se disabilitato (impostato su **no**), questi errori non vengono riportati. Questa opzione è abilitata per impostazione predefinita. La sua disabilitazione aumenta leggermente la velocità di esecuzione di Memcheck.

--ignore-ranges

Permette all'utente di specificare una o più gamme da ignorare durante il controllo sull'usabilità di parte della memoria per altri scopi. Gamme multiple sono delimitate da virgole, per esempio **--ignore-ranges=0xPP-0xQQ,0xRR-0xSS**.

Per un elenco di opzioni completo consultare la documentazione inclusa in **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**.

5.3.2. Come usare Cachegrind per una analisi sull'uso della memoria

Cachegrind simula l'interazione del programma con una gerarchia cache della macchina e (facoltativamente) il branch predictor. Controlla l'uso dell'istruzione simulata del primo-livello e le cache dei dati, per il rilevamento di una interazione del codice insufficiente con questo livello di cache; e la cache dell'ultimo livello, se risulta essere una cache di secondo o terzo livello, per un controllo dell'accesso per la gestione della memoria. Per questo i programmi eseguiti con Cachegrind avranno una velocità di esecuzione dai venti ai cento volte minore rispetto ad una loro esecuzione normale.

Per eseguire Cachegrind usare il seguente comando sostituendo *program* con il programma da usare con Cachegrind:

```
# valgrind --tool=cachegrind program
```

Cachegrind è in grado di ottenere le seguenti informazioni per l'intero programma e per ogni funzione presente al suo interno:

- Richieste di lettura non eseguite (read misses) e lettura (o istruzioni eseguite) delle istruzioni per cache del primo livello, e richieste di lettura non eseguite per l'istruzione della cache dell'ultimo livello;

- Lettura dati della cache (o lettura della memoria), richieste di lettura non eseguite e richieste di lettura non eseguite dei dati della cache dell'ultimo livello;
- Scrittura dati della cache (o scrittura della memoria), richieste di scrittura non eseguite e richieste di scrittura non eseguite dei dati della cache dell'ultimo livello;
- branch condizionali eseguiti e previsti incorrettamente; e
- branch indiritti eseguiti e previsti incorrettamente.

Per impostazione predefinita Cachegrind stampa un sommario delle informazioni relative alle statistiche sopra riportate, e scrive in modo più dettagliato le informazioni sul profiling su un file (**cachegrind.out.*pid***, dove *pid* è l'ID del processo del programma sul quale è stato eseguito Cachegrind). È possibile processare maggiormente questo file usando **cg_annotate** nel modo seguente:

```
# cg_annotate cachegrind.out.pid
```

NOTA

cg_annotate è in grado di riportare righe più lunghe di 120 caratteri in base alla lunghezza del percorso. Per rendere l'output più chiaro e facile da consultare, è consigliato aumentare la grandezza della finestra del terminale prima di eseguire il suddetto comando.

È possibile altresì confrontare i file del profilo creato da Cachegrind in modo da facilitare l'implementazione di un grafico delle prestazioni del programma prima e dopo una modifica. Per fare questo usare il comando **cg_diff**, sostituendo *first* con il file dell'output del profilo iniziale, e *second* con il file dell'output del profilo seguente:

```
# cg_diff first second
```

Questo comando produce un file dell'output combinato visualizzabile in modo dettagliato con **cg_annotate**.

Cachegrind supporta un numero di opzioni per il suo output. Alcune delle opzioni sono:

--I1

Specifica la dimensione, l'associazione e la dimensione della riga per la cache dell'istruzione di primo-livello, separati da virgole: **--I1=*size*,*associativity*,*line size***.

--D1

Specifica la dimensione, l'associazione e la dimensione della riga per la cache dei dati di primo-livello, separati da virgole: **--D1=*size*,*associativity*,*line size***.

--LL

Specifica la dimensione, l'associazione e la dimensione della riga della cache dell'ultimo-livello, separati da virgole: **--LL=*size*,*associativity*,*line size***.

--cache-sim

Abilita o disabilita la raccolta dei conteggi relativi al mancato accesso e dell'accesso della cache. Il valore predefinito è **yes** (abilitato).

Da notare che disabilitando entrambe le opzioni sopra indicate e **--branch-sim**, non permetterete a Cachegrind di ottenere alcuna informazione.

--branch-sim

Abilita o disabilita la raccolta delle informazioni relative ai conteggi previsti incorrettamente e sulle istruzioni del branch. Per impostazione predefinita questa opzione è impostata su **no** (disabilitato), poichè rallenterà Cachegrind di circa il 25 per-cento.

Da notare che disabilitando entrambe le opzioni sopra indicate e **--cache-sim**, non permetterete a Cachegrind di ottenere alcuna informazione.

Per un elenco di opzioni completo consultare la documentazione inclusa in **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**.

5.3.3. Profiling dello spazio di stack e heap con Massif

Massif misura lo spazio heap usato da un programma specifico; sia lo spazio utilizzabile che qualsiasi spazio aggiuntivo assegnato per processi di allineamento e contabilità. Il suo utilizzo aiuta a ridurre la quantità di memoria usata dal programma, al tempo stesso aumenta la velocità e riduce la possibilità di esaurire lo spazio di swap della macchina sulla quale viene eseguito. Massif fornisce altresi anche le informazioni sulle sezioni del programma responsabili all'assegnazione della memoria heap. I programmi eseguiti con Massif vengono eseguiti circa venti volte più lentamente rispetto alla loro velocità normale di esecuzione.

Per analizzare l'uso di heap da parte di un programma specificare **massif** come strumento di Valgrind da usare:

```
# valgrind --tool=massif program
```

I dati sul profiling raccolti da Massif vengono scritti su un file che per impostazione predefinita viene chiamato **massif.out.*pid***, dove *pid* è l'ID del processo del *programma* specificato.

Questi dati possono essere rappresentati in modo grafico con il comando **ms_print**:

```
# ms_print massif.out.pid
```

Così facendo verrà generato un grafico sul consumo della memoria relativa all'esecuzione del programma, e riportate le informazioni dettagliate sui siti responsabili per l'assegnazione in vari punti nel programma, incluso il punto massimo dell'assegnazione della memoria.

Massif fornisce un certo numero di opzioni della linea di comando utilizzabili. Alcune opzioni disponibili sono:

--heap

Specifica se eseguire il profiling di heap. Il valore predefinito è **yes**. Il profiling di heap può essere disabilitato impostando questa opzione su **no**.

--heap-admin

Specifica il numero di byte per blocco da usare per la gestione quando il profiling di heap è abilitato. Il valore predefinito è **8** byte per blocco.

--stacks

Specifica se eseguire il profiling dello stack. Il valore predefinito è **no** (disabilitato). Per abilitare il profiling dello stack impostare questa opzione su **yes**, ma attenzione poichè così facendo verrà sensibilmente rallentata l'esecuzione di Massif. Da notare altresì che Massif assume che la dimensione dello stack sia zero all'avvio, in modo da indicare meglio la dimensione della porzione di stack rispetto al quale il programma sul quale viene eseguito il profiling ha un controllo.

--time-unit

Specifica l'unità di tempo usata per il profiling. Sono disponibili tre valori validi per questa opzione: istruzioni eseguite (**i**), il valore predefinito, che risulta utile in numerosi casi; real time (**ms**, in millisecondi), utile in determinate istanze; e byte allocati/deallocati sull'heap/o stack (**B**), utile per programmi eseguiti brevemente e per l'esecuzione di test, poichè questo processo risulta essere quello più facilmente riproducibile su diverse macchine. Questa opzione è utile per un grafico dell'output di Massif con **ms_print**.

Per un elenco di opzioni completo consultare la documentazione inclusa in **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**.

5.4. OTTIMIZZAZIONE CAPACITÀ

Consultare questa sezione per informazioni sulla memoria, kernel e capacità del file system, per i parametri relativi ed i compromessi presenti nella modifica di questi parametri.

Per impostare temporaneamente questi valori durante il processo di ottimizzazione, usare il comando echo con il valore desiderato per il file appropriato nel file system proc. Per esempio per impostare ***overcommit_memory*** momentaneamente su **1**, eseguire:

```
# echo 1 > /proc/sys/vm/overcommit_memory
```

Da notare che il percorso per il parametro nel file system proc varia in base al sistema interessato alla modifica.

Per impostare questi valori in modo permanente, usare il comando **sysctl**. Per maggiori informazioni consultare la *Deployment Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

Valori ottimizzabili della memoria relativi alla capacità

Questi parametri sono posizionati in **/proc/sys/vm/** nel file system proc.

overcommit_memory

Definisce le condizioni che determinano se una richiesta di memoria molto grande può essere accettata o negata. Per questo parametro sono disponibili tre valori:

- **0** — L'impostazione predefinita. Il kernel esegue una gestione euristica del sovraccarico della memoria disponibile, negando le richieste chiaramente invalide. Sfortunatamente poichè la memoria viene assegnata usando un algoritmo euristico e non preciso, questa impostazione può talvolta sovraccaricare la memoria disponibile del sistema.
- **1** — Il kernel non esegue alcuna gestione del sovraccarico della memoria. Con questo tipo di impostazione si aumentano le possibilità di un sovraccarico della memoria, ma anche delle prestazioni per compiti "memory-intensive".
- **2** — Il kernel nega le richieste per una memoria uguale o maggiore alla somma totale tra lo

swap disponibile e la percentuale di RAM fisica specificata in ***overcommit_ratio***. Questa impostazione è ideale se si desidera avere un minor rischio di un utilizzo eccessivo della memoria.

NOTA

Questa impostazione è consigliata solo per sistemi con un'area di swap più grande della memoria fisica.

overcommit_ratio

Specifica la percentuale di RAM fisica considerata quando ***overcommit_memory*** viene impostato su **2**. Il valore predefinito è **50**.

max_map_count

Definisce il numero massimo di aree per la mappatura della memoria che un processo è in grado di usare. In numerosi casi il valore predefinito di **65530** è quello più appropriato. Aumentate questo valore se l'applicazione deve mappare un numer maggiore di file.

nr_hugepages

Definisce il numero di hugepages configurate nel kernel. Il valore predefinito è 0. È possibile solo allocare (o deallocare) hugepages se è presente un numero sufficiente di pagine fisiche disponibili adiacenti. Pagine riservate da questo parametro non possono essere usate per altri scopi. Maggiori informazioni sono disponibili nella documentazione installata: **/usr/share/doc/kernel-doc-*kernel_version*/Documentation/vm/hugetlbpage.txt**

Parametri regolabili del kernel relativi alla capacità

I parametri di seguito indicati sono posizionati in **/proc/sys/kernel/** nel file system proc.

msgmax

Definisce la dimensione massima permessa in byte di qualsiasi mesaggio in una coda. Questo valore non deve superare la dimensione della coda (***msgmnb***). Il valore predefinito è **65536**.

msgmnb

Definisce la dimensione massima in byte di una coda singola del messaggio. Il valore predefinito è **65536** byte.

msgmni

Definisce il numero massimo di identificatori della coda del messaggio (e quindi il numero massimo di code). Il valore predefinito su macchine con una architettura a 64-bit è **1985**; per architetture a 32-bit il valore predefinito è **1736**.

shmall

Definisce il numero totale di memoria condivisa in byte utilizzabile sul sistema in un dato momento. Il valore predefinito su macchine con una architettura a 64-bit è **4294967296**; per architetture a 32-bit il valore predefinito è **268435456**.

shmmax

Definisce il segmento massimo della memoria condivisa permesso dal kernel, in byte. Il valore predefinito sulle macchine con una architettura a 64-bit è **68719476736**; per architetture a 32-bit il valore predefinito è **4294967295**. Da notare tuttavia che il kernel supporta valori molto più grandi.

shmmni

Definisce il numero massimo per l'intero sistema di segmenti della memoria condivisa. Il valore predefinito è **4096** sia per l'architettura a 64-bit che per quella a 32-bit.

threads-max

Definisce il numero massimo di thread (compiti) per l'intero sistema utilizzabili dal kernel in un dato momento. Il valore predefinito è uguale al valore di ***max_threads*** del kernel. La formula usata è:

```
max_threads = mempages / (8 * THREAD_SIZE / PAGE_SIZE )
```

Il valore minimo di ***threads-max*** è **20**.

Parametri regolabili del file system relativi alla capacità

Questi parametri sono posizionati in **/proc/sys/fs/** nel file system proc.

aio-max-nr

Definisce il numero massimo di eventi permessi in tutti i contesti I/O asincroni attivi. Il valore predefinito è **65536**. Da notare che la modifica di questo valore non cambia la dimensione ne preassegna le strutture dei dati del kernel.

file-max

Elenca il numero massimo di operazioni di gestione dei file assegnati dal kernel. Il valore predefinito corrisponde al valore di ***files_stat.max_files*** nel kernel, ed è impostato prendendo in considerazione il valore più grande tra **(mempages * (PAGE_SIZE / 1024)) / 10** o **NR_FILE** (8192 in Red Hat Enterprise Linux). Aumentando questo valore potreste risolvere gli errori causati da un numero insufficiente di gestioni disponibili del file.

Parametri regolabili di Out-of-Memory Kill

Out of Memory (OOM) si riferisce ad uno stato dove tutta la memoria disponibile, incluso lo spazio di swap, è stata assegnata. Per default questa impostazione causa un panic del sistema ed un conseguente arresto delle sue funzioni. Tuttavia l'impostazione di ***/proc/sys/vm/panic_on_oom*** su **0** indica al kernel di invocare la funzione **oom_killer** in presenza di OOM. Generalmente **oom_killer** è in grado di arrestare "kill" processi fittizi, e permette al sistema di continuare con le sue funzioni.

Il seguente parametro può essere assegnato in base al processo e conferisce all'utente un maggiore controllo sui processi da arrestare usando la funzione **oom_killer**. Esso è posizionato in **/proc/*pid*/** nel file system proc, dove *pid* è il numero dell'ID del processo.

oom_adj

Definisce un valore da **-16** a **15** in grado di determinare l'***oom_score*** di un processo. Più alto è il valore di ***oom_score*** e maggiore è la possibilità che un processo venga arrestato "killed" da **oom_killer**. L'impostazione di ***oom_adj*** su **-17** disabilita **oom_killer** per quel processo.

IMPORTANTE

Ogni processo generato da un processo modificato erediterà l'***oom_score*** di quel processo. Per esempio, se un processo **sshd** è protetto dalla funzione **oom_killer**, tutti i processi iniziati da quella sessione SSH verranno protetti. Ciò può interessare l'abilità della funzione **oom_killer** di salvare il sistema in presenza di un OOM.

5.5. OTTIMIZZAZIONE DELLA MEMORIA VIRTUALE

La memoria virtuale viene generalmente usata dai processi, dalla cache del file system e dal kernel. Il suo utilizzo dipende da un certo numero di fattori i quali possono essere influenzati dai seguenti parametri:

swappiness

Un valore da 0 a 100 controlla il grado con il quale viene eseguito lo swap del sistema. Un valore elevato da una priorità alle prestazioni, ed esegue uno swap aggressivo dei processi dalla memoria fisica quando gli stessi non risultano attivi. Un valore basso conferisce una priorità ai processi di interazione ed evita lo swap dei processi dalla memoria fisica il più a lungo possibile. Il valore predefinito è **60**.

min_free_kbytes

Il numero minimo di kilobyte da mantenere libero su tutto il sistema. Questo valore viene usato per calcolare un valore limite per ogni zona con una memoria bassa, alla quale viene assegnata un numero di pagine libere riservate proporzionale alla dimensione.

AVVERTIMENTO

Fate attenzione quando impostate questo parametro poichè valori troppo bassi o valori troppo alti possono essere controproducenti.

Una impostazione troppo bassa di ***min_free_kbytes*** impedisce al sistema di riappropriarsi della memoria. Tale impostazione può risultare in una sospensione del sistema e in un arresto da parte di OOM di processi multipli.

Tuttavia l'impostazione di questo parametro ad un valore troppo alto (5-10% del totale della memoria del sistema) causerà l'uso immediato di tutta la memoria. Linux è stato creato per utilizzare tutta la RAM disponibile per memorizzare in cache i dati del file system. Impostando un valore ***min_free_kbytes*** elevato causerà l'impiego di un periodo più lungo da parte del sistema per riappropriarsi della memoria.

dirty_ratio

Definisce un valore percentuale. La rimozione dei dati corrotti (tramite **pdflush**) inizia quando i dati raggiungono questa percentuale del totale della memoria del sistema. Il valore predefinito è **20**.

dirty_background_ratio

Definisce un valore percentuale. La rimozione dei dati corrotti (tramite **pdflush**) inizia nello sfondo quando i dati raggiungono questa percentuale del totale della memoria del sistema. Il valore predefinito è **10**.

drop_caches

L'impostazione di questo valore su **1**, **2**, o **3** causa l'abbandono di varie combinazioni da parte del kernel di cache slab e pagine.

1

Il sistema non convalida e libera tutta la memoria in cache della pagina.

2

Il sistema libera tutta la memoria in cache della slab non utilizzata.

3

Il sistema libera tutta la cache della pagina e la memoria in cache della slab.

Questa è una operazione non-distruttiva. Poichè non è possibile rendere disponibili oggetti corrotti, è consigliata l'esecuzione di **sync** prima di impostare il valore di questo parametro.

IMPORTANTE

Non è consigliato l'uso di ***drop_caches*** per liberare la memoria in un ambiente di produzione.

Per impostare temporaneamente questi valori durante il processo di ottimizzazione, usare il comando echo con il valore desiderato per il file appropriato nel file system proc. Per esempio per impostare ***swappiness*** momentaneamente su **50**, eseguire:

```
# echo 50 > /proc/sys/vm/swappiness
```

Per impostare questo valore in modo permanente, usare il comando **sysctl**. Per maggiori informazioni consultare la *Deployment Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

CAPITOLO 6. INPUT/OUTPUT

6.1. FUNZIONI

Red Hat Enterprise Linux 6 introduce un certo numero di miglioramenti nello stack I/O.

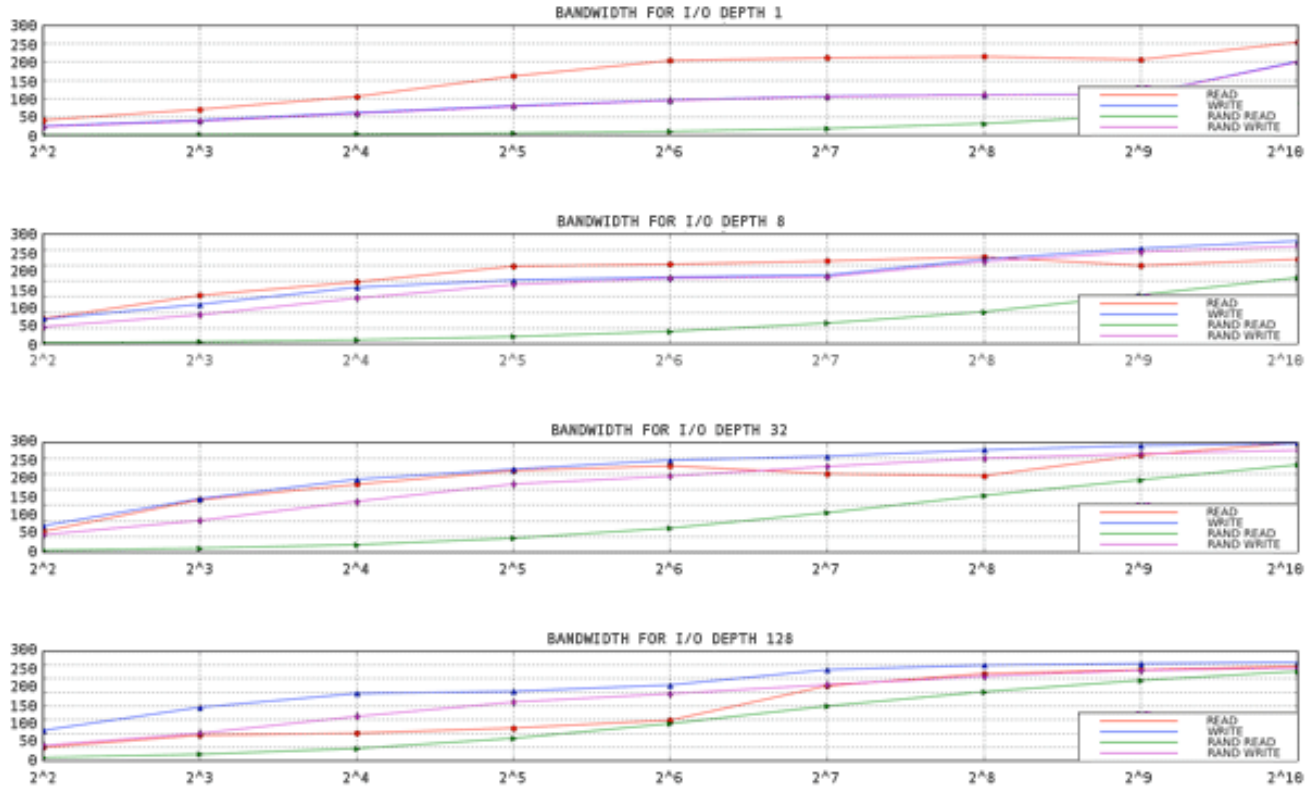
- I Solid state disk (SSD) sono ora riconosciuti automaticamente e le prestazioni dello scheduler I/O sono ottimizzate per trarre vantaggio dal valore I/Os per second (IOPS) elevato di questi dispositivi.
- È stato aggiunto al kernel il supporto per il Discard per notificare allo storage sottostante la gamma di blocchi non usati. Ciò assiste il SSD durante l'uso degli algoritmi wear-leveling e assiste lo storage che supporta il provisioning del blocco logico (una forma di spazio dell'indirizzo virtuale per lo storage) controllando in modo più dettagliato la quantità di storage in uso.
- Con l'introduzione di Red Hat Enterprise Linux 6.1 l'implementazione dei barrier per il file system è stata rivista in modo da avere migliori prestazioni.
- **`pdflush`** è stato sostituito dai per-backing-device flusher thread, i quali migliorano sensibilmente la scalabilità del sistema in configurazioni con conteggi LUN molto grandi.

6.2. ANALISI

Una ottimizzazione corretta delle prestazioni per lo stack dello storage richiede una comprensione dei flussi dati nel sistema e dello storage sottostante, insieme al suo comportamento in presenza di diversi carichi di lavoro. Tale processo necessita altresì di una conoscenza del carico di lavoro da ottimizzare.

Quando implementate un nuovo sistema è consigliato tracciare un profilo dello storage dal basso verso l'alto. Iniziate con i dischi o LUN raw ed esaminatene le prestazioni usando un I/O diretto (I/O in grado di baipassare la cache della pagina del kernel). Questo è il test più semplice e rappresenterà lo standard per mezzo del quale verranno misurate le prestazioni I/O all'interno dello stack. Iniziate con un generatore del carico di lavoro di base (come ad esempio **aio-stress**) in grado di produrre processi di lettura e scrittura randomici sequenziali su una gamma di dimensioni I/O e code.

Di seguito viene riportato un grafico di una serie di esecuzioni **aio-stress**, ognuna delle quali esegue quattro fasi: scrittura sequenziale, lettura sequenziale, scrittura randomica e lettura randomica. In questo esempio lo strumento viene configurato per una esecuzione su una gamma di dimensioni (x axis) e profondità della coda (una per grafico). La profondità della coda rappresenta il numero totale di operazioni I/O in corso in un dato momento.



y-axis mostra la larghezza di banda in megabyte per secondo. x-axis mostra la dimensione I/O in kilobyte.

Figura 6.1. output aio-stress per 1 thread, 1 file

Da notare come la stringa dell'output netto vada dall'angolo sinistro in basso a quello destro in alto. Altresì, per una dimensione data, è possibile ottenere un output netto aggiuntivo dallo storage tramite l'aumento del numero di I/O in corso.

Eseguendo questi carichi di lavoro semplici con lo storage, sarà possibile comprendere le varie prestazioni dello storage stesso con un determinato carico di lavoro. Conservate i dati generati dai test per un confronto con carichi di lavoro più complessi.

Se desiderate utilizzare il device mapper o md, aggiungetelo e ripetete i test. Se notate una perdita sensibile di prestazioni assicuratevi che sia prevista e che possa essere spiegata. Per esempio, una perdita di prestazioni può essere prevista se aggiungete nello stack un livello raid di checksum. Al contrario una perdita di prestazioni non prevista può essere causata da operazioni I/O non allineate. Per impostazione predefinita Red Hat Enterprise Linux è in grado di allineare le partizioni e i metadati del device mapper in modo ottimale. Tuttavia non tutti i tipi di storage riportano il proprio allineamento ottimale e per questo motivo potrà essere necessaria una ottimizzazione manuale.

Dopo aver aggiunto un device mapper o md aggiungete un file system sopra al dispositivo a blocchi ed eseguitene il test usando un I/O diretto. Ancora una volta confrontate i risultati con i test precedenti ed assicuratevi di capire eventuali discrepanze. Direct-write I/O generalmente ha una migliore prestazione su file preassegnati, per questo motivo assicuratevi di preassegnare i file prima di eseguire una prova delle prestazioni.

Generatori del carico di lavoro sintetici utili includono:

- **aio-stress**
- **iozone**
- **fio**

6.3. STRUMENTI

Sono disponibili un certo numero di strumenti per la diagnosi dei problimi relativi alle prestazioni nei sottosistemi I/O. **vmstat** fornisce una breve panoramica sulle prestazioni del sistema. Le seguenti colonne riportano i parametri più importanti per I/O: ***si*** (swap in), ***so*** (swap out), ***bi*** (block in), ***bo*** (block out), e ***wa*** (I/O wait time). ***si*** e ***so*** sono utili quando lo spazio di swap è sullo stesso dispositivo della partizione dati, e risulta indicativo della pressione generale della memoria. ***si*** e ***bi*** sono operazioni di lettura mentre ***so*** e ***bo*** sono operazioni di scrittura. Queste categorie sono riportate in kilobyte. ***wa*** è il tempo di inattività; Esso indica quale sezione della coda di esecuzione è stata bloccata in attesa del completamento dell'I/O.

Analizzando il sistema con **vmstat** potrete scoprire se il sottosistema I/O sia il responsabile per eventuali problemi di prestazione. A tal proposito vale la pena consultare anche le colonne ***free***, ***buff***, e ***cache*** . L'aumento del valore ***cache*** insieme a ***bo*** seguito da una riduzione di ***cache*** ed un aumento di ***free***, indica che il sistema esegue una operazione di write-back e di annullamento della cache della pagina.

Da notare come i numeri di I/O riportati da **vmstat** risultano essere aggregazioni di tutti gli I/O per tutti i dispositivi. Una volta determinata la presenza di un abbassamento delle prestazioni nel sottosistema I/O, sarà possibile esaminare il problema in dettaglio con **iostat**, il quale eseguirà una analisi dettagliata del riporto I/O in base al dispositivo. È possibile altresì ripristinare informazioni più dettagliate, come ad esempio la dimensione media della richiesta, il numero di processi di scrittura e scrittura per secondo e la quantità di operazioni Merge dell'I/O in corso.

Tramite la dimensione media della richiesta e la dimensione media della coda (**avgqu-sz**), sarà possibile stimare le prestazioni dello storage usando i grafici generati durante l'analisi delle prestazioni. Sono applicate alcune generalizzazioni, se la dimensione media delle richieste è 4KB e quella della coda risulta essere 1, l'output netto non dovrebbe risultare performante.

Se i valori della prestazione non corrispondono alla prestazione prevista, sarà possibile eseguire una analisi più dettagliata con **blktrace**. La suite di utilità **blktrace** rende disponibili informazioni più dettagliate sul tempo trascorso nel sottosistema I/O. L'output di **blktrace** è un insieme di file di traccia binari che possono essere processati in un secondo momento da altre utilità come ad esempio **blkparse**.

blkparse è una utilità usata insieme a **blktrace**. Essa legge l'output dalla traccia e produce una breve versione di testo.

Il seguente è un esempio di output **blktrace**:

```
8,64   3        1     0.000000000  4162  Q  RM 73992 + 8 [fs_mark]
8,64   3        0     0.000012707     0  m   N cfq4162S / alloced
8,64   3        2     0.000013433  4162  G  RM 73992 + 8 [fs_mark]
8,64   3        3     0.000015813  4162  P   N [fs_mark]
8,64   3        4     0.000017347  4162  I   R 73992 + 8 [fs_mark]
8,64   3        0     0.000018632     0  m   N cfq4162S / insert_request
8,64   3        0     0.000019655     0  m   N cfq4162S / add_to_rr
8,64   3        0     0.000021945     0  m   N cfq4162S / idle=0
8,64   3        5     0.000023460  4162  U   N [fs_mark] 1
8,64   3        0     0.000025761     0  m   N cfq workload slice:300
8,64   3        0     0.000027137     0  m   N cfq4162S / set_active
wl_prio:0 wl_type:2
8,64   3        0     0.000028588     0  m   N cfq4162S / fifo=(null)
8,64   3        0     0.000029468     0  m   N cfq4162S / dispatch_insert
8,64   3        0     0.000031359     0  m   N cfq4162S / dispatched a
request
```

```
8,64   3        0     0.000032306     0  m   N cfq4162S / activate rq,
drv=1
8,64   3        6     0.000032735  4162  D   R 73992 + 8 [fs_mark]
8,64   1        1     0.004276637     0  C   R 73992 + 8 [0]
```

L'output è complesso e difficile da laggere. È possibile sapere i processi responsabili per l'I/O sul dispositivo, quale risulta essere utile, ma **blkparse** è in grado di rendere disponibili maggiori informazioni in un formato più semplice. Le informazioni di **blkparse** sono riportate nella parte finale di questo output:

```
Total (sde):
Reads Queued:          19,       76KiB  Writes Queued:     142,183,
568,732KiB
Read Dispatches:       19,       76KiB  Write Dispatches:   25,440,
568,732KiB
Reads Requeued:         0               Writes Requeued:       125
Reads Completed:       19,       76KiB  Writes Completed:   25,315,
568,732KiB
Read Merges:            0,        0KiB  Write Merges:      116,868,
467,472KiB
IO unplugs:        20,087               Timer unplugs:           0
```

Il sommario mostra il tasso di I/O medio, l'attività di merge e confronta i carichi di lavoro delle operazioni di lettura con quelle di scrittura. Per la maggior parte dei casi tuttavia l'output di **blkparse** è troppo ampio per essere utile da solo. Fortunatamente sono disponibili alcuni strumenti per la visualizzazione dei dati più importanti.

btt fornisce una analisi della quantità di tempo che l'I/O ha trascorso nelle diverse aree dello stack. Queste aree sono:

- Q – Un blocco I/O messo in coda
- G – Ottieni una richiesta

  Un blocco I/O appena messo in coda non è il condidato per una opearazione di Merge con una richiesta esistente, per questo motivo è stato assegnata una nuova richiesta del livello del blocco.
- M – Un blocco I/O è stato unito con una richiesta esistente.
- I – Inserita una richiesta nella coda del dispositivo.
- D – Emessa una richiesta al dispositivo.
- C – Richiesta completata dall'unità.
- P – La coda del dispositivo è bloccata "Plugged" e permette la raccolta di richieste.
- U – La coda del dispositivo è sbloccata "Unplugged", e permette l'emissione delle richieste per il dispositivo.

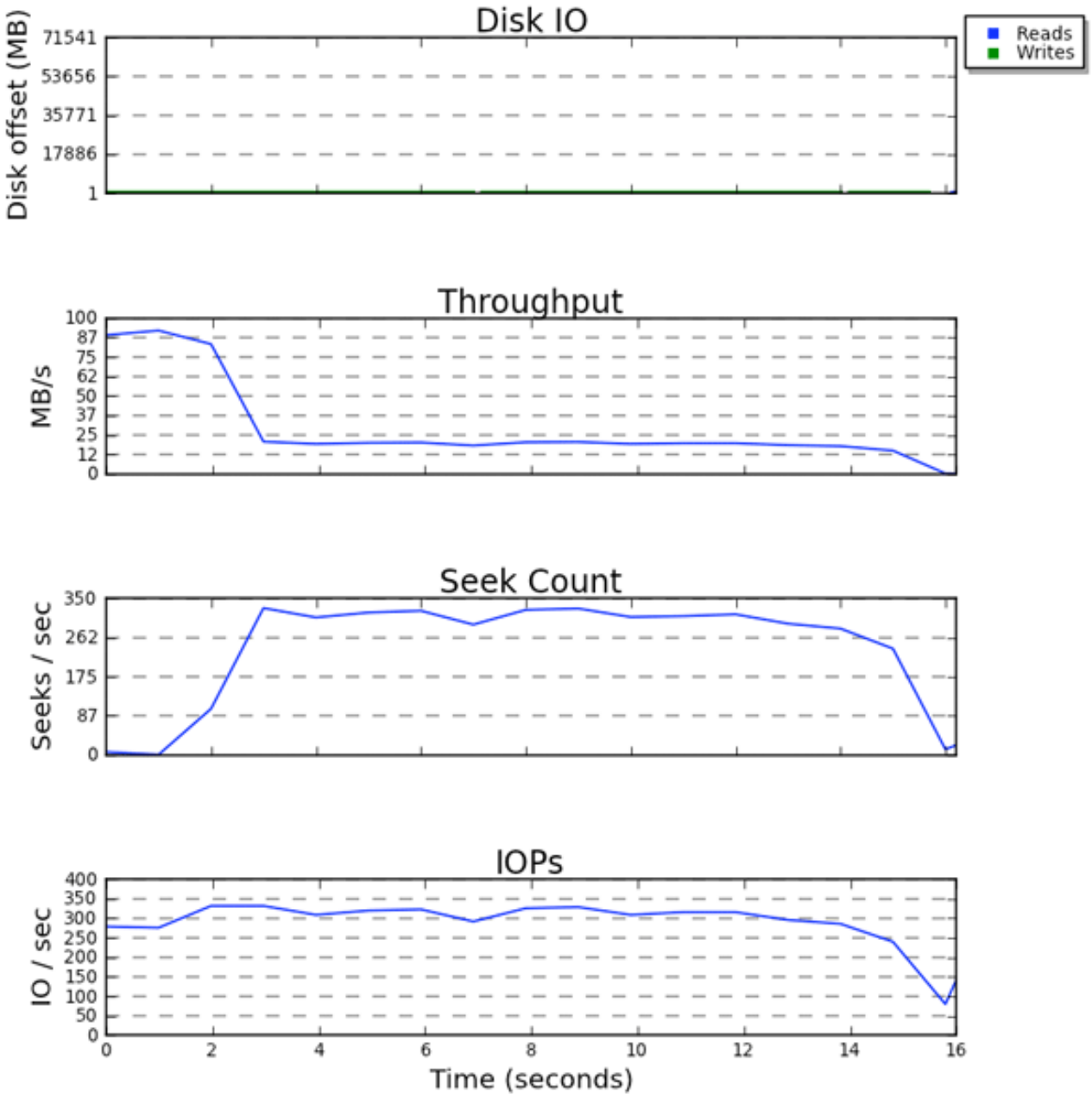
btt suddivide in modo dettagliato il tempo trascorso in ogni area e quello usato durante la transizione tra le aree, ad esempio:

- Q2Q – tempo trascorso tra richieste inviate al livello blocco

- Q2G — il tempo trascorso quando un blocco I/O viene messo in coda ed il tempo nel quale riceve una richiesta
- Q2I — il tempo trascorso da quando una richiesta è stata assegnata a quando la stessa viene inserita nella coda del dispositivo
- Q2M — il tempo trascorso quando un blocco I/O viene messo in coda ed il tempo entro il quale viene unito con una richiesta esistente
- Q2I — tempo trascorso quando una richiesta viene inserita nella coda del dispositivo ed il tempo entro il quale la richiesta viene inviata al dispositivo
- Q2D — il tempo trascorso entro il quale un blocco I/O viene unito con una richiesta esistente fino a quando la richiesta viene emessa al dispositivo
- D2C — tempo di servizio della richiesta per il dispositivo
- Q2C — tempo totale trascorso nel blocco per un richiesta

Da questa tabella è possibile avere numerose informazioni sul carico di lavoro. Per esempio se Q2Q è più grande di Q2C, ciò significa che l'applicazione non emette alcun I/O in rapida successione. Quindi qualsiasi problema relativo alle prestazioni potrebbe non avere alcuna relazione con il sottosistema I/O. Se D2C è molto alto il dispositivo necessita di un tempo più lungo per rispondere alle richieste. Tale comportamente potrebbe indicare che il dispositivo è semplicemente sovraccarico (ciò potrebbe derivare dal fatto che esso risulta essere una risorsa condivisa), oppure poichè il carico di lavoro inviato al dispositivo non è ottimale. Se Q2G è molto elevato ciò indicherà un numero molto grande di richieste messe in coda contemporaneamente. Ciò indicherà che lo storage non è in grado di gestire il carico I/O.

Per finire **seekwatcher** utilizza dati binari **blktrace** e genera un insieme di grafici, incluso Logical Block Address (LBA), output netti, ricerche per secondo, e I/Os Per Second (IOPS).



Avg Seeks/s	Avg MB/s	Avg IO/s	Run time (s)
252.57	31.72	305.99	16.21

Figura 6.2. Esempio di output di seekwatcher

Tutti i grafici riportano il tempo con l'asse X. Il grafico LBA mostra i processi di lettura e scrittura in diversi colori. Da notare la relazione che intercorre tra i grafici dell'output netto e di ricerca/sec. Per uno storage sensibile alle ricerche, vi è una relazione inversa tra i due grafici. Il grafico IOPS è utile se per esempio non ricevete l'output netto previsto da un dispositivo, ma al tempo stesso raggiungete i limiti IOPS.

6.4. CONFIGURAZIONE

Una delle prime decisioni da prendere è il tipo di scheduler I/O da usare. Questa sezione fornisce una panoramica su ogni scheduler per assistere l'utente ad implementare quello più idoneo al carico di lavoro desiderato.

6.4.1. Completely Fair Queuing (CFQ)

Il CFQ cerca di essere il più imparziale possibile nelle decisioni di programmazione dell'I/O prendendo in considerazione il processo che ha iniziato l'I/O. Sono disponibili tre tipi di classi di programmazione: real-time (RT), best-effort (BE), e idle. È possibile assegnare manualmente una classe di programmazione ad un processo tramite il comando **`ionice`** o attraverso la chiamata del sistema **`ioprio_set`**. Per impostazione predefinita i processi vengono posizionati nella classe best-effort (BE). Le classi real-time (RT) e best-effort vengono suddivise in otto priorità I/O all'interno di ogni classe, con una priorità 0 corrispondente al valore più alto e 7 a quello più basso. I processi nella classe real-time vengono programmati in modo più aggressivo rispetto a quelli presenti in best-effort o idle, per questo motivo ogni I/O real-time programmato viene sempre eseguito prima della classe best-effort o idle. Ciò significa che la priorità real-time può annullare sia la classe best-effort che idle. La classe di programmazione Best effort è quella predefinita e 4 è la priorità di default. I processi nella classe idle vengono serviti solo quando non vi è alcun I/O in attesa nel sistema. Per questo è importante impostare solo la classe di programmazione di un processo su idle se l'I/O non è necessario per procedere con il normale funzionamento.

CFQ garantisce una imparzialità fornendo un periodo di tempo ad ogni processo che esegue un I/O. Durante questo periodo un processo può avere (per impostazione predefinita) fino a 8 richieste per volta. Uno scheduler cercherà di anticipare se una applicazione emetterà un numero maggiore di I/O prendendo in considerazione gli eventi passati. Se è previsto che un processo emetterà un numero maggiore di I/O, CFQ assumerà uno stato di attesa per l'I/O specifico anche se altri I/O sono in attesa di essere emessi.

A causa della sospensione eseguita da CFQ, questa impostazione non sempre risulta essere la più idonea per hardware senza problemi molto grossi di ricerca, come ad esempio array di storage esterni o dischi di tipo solid state. Se è necessario utilizzare CFQ su storage simili (per esempio, se desiderate anche utilizzare il cgroup proportional weight I/O scheduler), sarà necessario ottimizzare alcune impostazioni per migliorare le prestazioni di CFQ. Impostare i seguenti parametri nei file con lo stesso nome presenti in **`/sys/block/`*`device`*`/queue/iosched/`**:

```
slice_idle = 0
quantum = 64
group_idle = 1
```

Quando ***`group_idle`*** viene impostato su 1, sarà ancora possibile il verificarsi di uno stallo degli I/O (per cui lo storage backend non risulta occupato a causa di uno stato inattivo). Tuttavia esso avrà una frequenza ridotta rispetto agli stati di inattività presenti su ogni coda del sistema.

Un CFQ è uno scheduler I/O di tipo non-work-conserving, esso infatti può entrare in sospensione in presenza di richieste in attesa (come affrontato precedentemente). Lo stack di scheduler di tipo non-work-conserving può introdurre una latenza superiore nel percorso I/O. Un esempio di stack simile è l'uso di CFQ sui controller RAID hardware basati sull'host. I controller RAID possono implementare i propri scheduler non-work-conserving , causando così ritardi su due livelli dello stack. Gli scheduler di tipo non-work-conserving operano nel modo migliore in presenza di un flusso elevato di dati sul quale basare le proprie decisione. In presenza di uno stack di questi algoritmi, lo scheduler nella posizione più bassa sarà in grado di vedere solo l'attività passata dallo scheduler posizionato sopra ad esso. Per questo motivo il livello inferiore potrà solo visualizzare un percorso I/O parziale che non rappresenta affatto il carico di lavoro effettivo.

Parametri ottimizzabili

back_seek_max

Le Backward seek sono processi dannosi per le prestazioni in quanto essi possono avere ritardi maggiori nel riposizionamento delle testine rispetto alle forward seek. Tuttavia CFQ è in grado di eseguirle se risultano essere piccole operazioni di ricerca. Questo parametro controlla la distanza massima abilitato dallo scheduler I/O per questo tipo di ricerche. Il valore predefinito è **16** KB.

back_seek_penalty

A causa della scarsa efficienza delle backward seek verrà loro associata una penalità. La penalità risulta essere un moltiplicatore; per esempio, considerate la posizione di una testina del disco su 1024KB. Supponiamo la presenza in coda di due richieste, una in 1008KB e l'altra in 1040KB. Le due richieste sono equidistanti dalla posizione corrente della testina. Tuttavia dopo aver applicato la penalità dovuta alla backward seek (default:2), la richiesta nella seconda posizione si trova ora due volte più vicina rispetto alla prima richiesta. Per questo motivo la testina si muoverà in avanti.

fifo_expire_async

Questo parametro controlla il periodo entro il quale una richiesta (scrittura in buffer) asincrona può non essere servita. Dopo questo periodo (in millisecondi) una richiesta asincrona non servita verrà spostata nell'elenco Invio. L'impostazione predefinita è **250** ms.

fifo_expire_sync

Simile al parametro ifo_expire_async, per richieste sincrone (lettura e scrittura O_DIRECT). Il valore predefinito è **125** ms.

group_idle

Quando impostato, CFQ sarà inattivo sull'ultimo processo che emette un I/O in un cgroup. Impostare questo valore su **1** quando utilizzate i proportional weight I/O cgroup ed impostare ***slice_idle*** su **0** (generalmente impostato su storage veloci).

group_isolation

Se group isolation è stato abilitato (su **1**), esso fornisce un isolamento più forte tra gruppi a scapito dell'output netto. In generale se group isolation è stato disabilitato, l'imparzialità viene fornita solo per carichi di lavoro sequenziali. Abilitando questo parametro avrete una imparzialità sia per carichi di lavoro randomici che per quelli sequenziali. Il valore predefinito è **0** (disabilitato). Per maggiori informazioni consultate **Documentation/cgroups/blkio-controller.txt**.

low_latency

Se il parametro low_latency è stato abilitato (impostato su **1**), CFQ cerca di fornire un tempo di attesa massimo di 300 ms per ogni processo che emette un I/O su un dispositivo. Questa impostazione assicura una certa imparzialità sull'output netto. Disabilitando questo parametro (impostandolo su **0**) verrà ignorata la latenza target, permettendo così ad ogni processo presente nel sistema di ricevere una quota di tempo completa. Per impostazione predefinita Low latency risulta essere abilitato.

quantum

Il quantum controlla il numero di I/O che il CFQ invierà allo storage in un determinato periodo, limitando così la profondità della coda del dispositivo. Per impostazione predefinita questo valore è impostato su **8**. Lo storage è in grado di supportare una coda più grande, ma se aumentate il valore del parametro ***quantum***, potrete impattare negativamente sulla latenza in particolare in presenza di una quantità molto elevata di processi di scrittura sequenziali.

slice_async

Questo valore controlla la quantità di tempo assegnata ad ogni processo che emette I/O asincroni (scrittura con buffer). Per impostazione predefinita questo valore è impostato su **40** ms.

slice_idle

Questo valore specifica la quantità di tempo che CFQ deve essere inattivo in attesa di altre richieste. Il valore predefinito in Red Hat Enterprise Linux 6.1, e versioni precedenti, è di **8** ms. Con Red Hat Enterprise Linux 6.2 e versioni più recenti questo valore è di **0**. Il valore zero migliora l'output netto di uno storage RAID esterno, rimuovendo tutti gli stati inattivi a livello albero del servizio e coda. Tuttavia questo valore può impattare negativamente sulle prestazioni di storage non-RAID interni poichè esso aumenta il numero generale di ricerche. Per storage non-RAID è consigliato un valore ***slice_idle*** maggiore di 0.

slice_sync

Questo valore controlla la quantità di tempo assegnata ad un processo che emette I/O sincroni (scrittura diretta o lettura). Per impostazione predefinita questo valore è impostato su **100** ms.

6.4.2. Deadline I/O Scheduler

Il deadline I/O scheduler cerca di conferire una latenza garantita alle richieste. È importante notare come la misurazione della latenza inizia solo quando le richieste arrivano allo scheduler I/O (distinzione importante poichè una applicazione può essere sospesa "sleep" per liberare i descrittori di richieste). Per impostazione predefinita i processi di lettura hanno priorità sui processi di scrittura, poichè le applicazioni possono bloccarsi con più frequenza in processi di lettura di I/O

Il deadline invia gli I/O in blocco o batch. Un blocco è una sequenza di I/O di scrittura o lettura con un ordine LBA crescente. Dopo la processazione di ogni blocco lo scheduler I/O controlla se i processi di scrittura sono sprovvisti di risorse, successivamente deciderà se iniziare un nuovo blocco di processi di lettura o scrittura. L'elenco FIFO di richieste verrà controllato solo per la presenza di richieste scadute per ogni blocco e successivamente per la direzione dei dati per il blocco stesso. Quindi se un blocco di processi di scrittura viene selezionato e al suo interno è presente un processo di lettura scaduto, quella richiesta non verrà servita fino a quando non verrà completato il blocco dei processi di scrittura.

Parametri ottimizzabili

fifo_batch

Questo valore determina il numero di processi di lettura o scrittura da emettere in un blocco singolo. Il valore predefinito è **16**. L'impostazione di un valore più alto può generare un miglior output netto, aumentando però al tempo stesso anche il valore di latenza.

front_merges

Impostare questo parametro su **0** se il carico di lavoro non eseguirà mai operazioni di front merge. Se non conoscete il sovraccarico presente con questa operazione, è consigliato lasciare il valore predefinito (**1**).

read_expire

Questo valore permette all'utente di impostare il numero di millisecondi nel quale una richiesta di lettura deve essere servita. Per impostazione predefinita questo valore è impostato su **500** ms (mezzo secondo).

write_expire

Questo valore permette all'utente di impostare il numero di millisecondi nel quale una richiesta di scrittura deve essere servita. Per impostazione predefinita questo valore è impostato su **5000** ms (cinque secondi).

writes_starved

Questo valore determina il numero di blocchi di lettura da processare prima di processare un blocco singolo di processi di scrittura. Più questo valore risulta essere alto e maggiore è la preferenza data ai processi di lettura.

6.4.3. Noop

Lo scheduler Noop I/O implementa un semplice algoritmo di programmazione first-in first-out (FIFO). Il merge delle richieste si verifica su un livello del blocco generico, ma risulta essere un last-hit cache. Se un sistema è stato associato ad una CPU e lo storage risulta essere veloce, questo tipo di scheduler I/O potrebbe essere quello migliore da usare.

Di seguito vengono riportati i parametri disponibili per il livello del blocco.

/sys/block/sdX/queue tunables

add_random

Spesso il sovraccarico degli eventi I/O che contribuiscono al bacino di entropia per **/dev/random** è misurabile. In tal caso è consigliato impostare questo valore su 0.

max_sectors_kb

Per impostazione predefinita la dimensione massima della richiesta inviata al disco è **512** KB. Questo parametro può essere usato per aumentare o diminuire quel valore. Il valore minimo è limitato dalla dimensione del blocco logico; il valore massimo è limitato da ***max_hw_sectors_kb***. Sono presenti alcuni SSD con prestazioni peggiori in presenza di dimensioni I/O maggiori rispetto alla dimensione dell'erase block size interno. In questi casi è consigliato diminuire ***max_hw_sectors_kb*** alla dimensione dell'erase block size. Per eseguire una prova utilizzate un generatore di I/O come ad esempio **iozone** o **aio-stress**, variando la dimensione, per esempio da **512** byte a **1** MB.

nomerges

Questo parametro è principalmente usato come ausilio al debugging. Numerosi carichi di lavoro traggono beneficio dal processo di unione delle richieste (anche su storage più veloci come SSD). In alcuni casi è tuttavia consigliato disabilitare il processo di unione "merging', per esempio se desiderate sapere quanti IOPS uno storage backend è in grado di processare senza disabilitare read-ahead o eseguire un I/O randomico.

nr_requests

Ogni coda ha un limite sul numero totale di descrittori di richieste assegnabili per ogni I/O di scrittura e lettura. Per impostazione predefinita questo valore è **128**, ciò significa che 128 letture e 128 scritture possono essere messi in coda contemporaneamente prima di sospendere "sleep" un processo. Il processo sospeso risulta essere quello che succesivamente cercherà di assegnare una richiesta, e potrà non essere il processo che ha assegnato tutte le richieste disponibili.

Se siete in possesso di una applicazione particolarmente sensibile alla latenza allora è consigliato ridurre il valore di ***nr_requests*** nella coda delle richieste, limitando al tempo stesso la profondità

della coda del comando sullo storage ad un numero più basso (anche usando un valore pari a **1**), in questo modo il writeback I/O non sarà in grado di assegnare tutti i descrittori di richiesta disponibili e consumare la coda del dispositivo con I/O di scrittura. Una volta assegnato un ***nr_requests***, tutti gli altri processi che cercano di eseguire un I/O verranno sospesi e messi in attesa di richieste disponibili. Questa impostazione è più imparziale poichè le richieste sono distribuite con una modalità round-robin (impedendo così ad un processo di consumare tutto in rapida successione). Da notare che questo è problematico solo in presenza di scheduler deadline o noop, poichè la configurazione predefinita di CFQ impedisce questo tipo di situazione.

optimal_io_size

In alcune circostanze lo storage sottostante riporterà una dimensione dell'I/O ottimale. Ciò è comune in presenza di hardware e software RAID, dove la dimensione I/O ottimale è quella del segmento. In presenza di questo valore, quando possibile, le applicazioni dovranno emettere un I/O allineato, e multiplo, della dimensione I/O ottimale.

read_ahead_kb

Il sistema operativo è in grado di rilevare quando l'applicazione esegue un processo di lettura sequenziale dei dati da un file o da un disco. Esso sarà in grado in questi casi di eseguire un algoritmo read-ahead intelligente, e leggere dal disco un numero maggiore di dati rispetto a quelli richiesti dall'utente. Quindi, alla lettura successiva di un blocco dati da parte dell'utente, essi saranno già presenti nella cache del sistema operativo. Un possibile effetto negativo di questa operazione è la lettura di un numero maggiore di dati da parte del sistema operativo rispetto a quelli necessari. Questa operazione occuperà spazio in cache fino a quando i dati verranno rimossi a causa di un uso troppo elevato della memoria. In circostanze simili, ed in presenza di processi multipli che eseguono operazioni read-ahead false, verrà aumentata la pressione dovuta ad un elevato uso della memoria.

Per dispositivi device mapper è consigliato aumentare il valore di ***read_ahead_kb***, ad esempio **8192**. Questa operazione è dovuta a causa del numero di dispositivi multipli sottostanti che compongono un device mapper. Per una corretta ottimizzazione impostate il valore predefinito (**128** KB) e moltiplicatelo per il numero dei dispositivi da mappare.

rotational

Tradizionalmente i dischi fissi sono di tipo "rotational" (costituiti da dischi rotanti). Tuttavia SSD non riflette questa tendenza e riporta questo tipo di impostazione in modo corretto. Se siete in presenza di un dispositivo il quale non indica con correttezza questo flag, impostate manualmente il parametro rotational su **0**; se il suddetto parametro è disabilitato, l'elevatore I/O non utilizza una logica per la riduzione di operazioni di ricerca, poichè per queste operazioni è presente una penalità ridotta su dispositivi "non-rotational".

rq_affinity

Il completamento degli I/O può essere effettuato su CPU diverse da quella che ha emesso l'I/O. Impostando ***rq_affinity*** su **1** il kernel invierà queste operazioni alla CPU che ha originato l'I/O. Questa operazione migliorerà l'efficacia di una archiviazione in cache dei dati della CPU.

CAPITOLO 7. FILE SYSTEM

Consultare questo capitolo per una panoramica sui file system supportati con Red Hat Enterprise Linux e su come ottimizzarne le loro prestazioni.

7.1. CONSIDERAZIONI SUL PROCESSO DI OTTIMIZZAZIONE PER I FILE SYSTEM

Sono presenti diversi fattori comuni a tutti i file system da considerare: opzioni di montaggio e formattazione selezionati e azioni disponibili alle applicazioni in grado di migliorare le prestazioni di un dato sistema.

7.1.1. Opzioni di formattazione

Dimensione blocco del file system

La dimensione dei blocchi può essere selezionata al momento del `mkfs`. La gamma delle dimensioni disponibili dipende dal sistema: il limite più alto è la dimensione massima della pagina del sistema host, mentre il limite più basso dipende dal file system usato. La dimensione del blocco predefinita è appropriata per la maggior parte dei casi.

Se desiderate creare numerosi file più piccoli rispetto alla dimensione del blocco predefinito, sarà possibile impostare una dimensione del blocco più piccolo per minimizzare la quantità di spazio usato sul disco. Da notare tuttavia che l'impostazione di una dimensione più piccola potrebbe limitare la dimensione massima del file system, e generare così un sovraccarico aggiuntivo, in particolare per i file più grandi, rispetto alla dimensione del blocco selezionato.

Geometria del file systema

Se il sistema utilizza uno storage segmentato, ad esempio RAID5, sarà possibile migliorare le prestazioni allineando i dati e i metadati con la geometria dello storage sottostante al momento del `mkfs`. Per RAID software ((LVM o MD) ed alcuni storage hardware di livello enterprise, queste informazioni vengono richieste ed impostate automaticamente, ma in numerosi casi l'amministratore dovrà specificare la geometria manualmente con `mkfs` sulla linea di comando.

Consultate la *Storage Administration Guide* per maggiori informazioni sulla creazione e la gestione dei file system.

Journal esterni

Con carichi di lavoro dei metadati-intensivi la sezione dei log di un file system con journal (come ad esempio ext4 e XFS) viene aggiornata molto spesso. Per minimizzare il tempo di ricerca "seek time" dal file system al journal, posizionare il journal su uno storage apposito. Da notare tuttavia che il posizionamento del journal su uno storage esterno più lento del file system primario, potrebbe annullare qualsiasi vantaggio associato con l'uso di uno storage di questo tipo.

AVVERTIMENTO

Assicuratevi che il journal esterno sia affidabile. La perdita di un dispositivo journal esterno causerà la corruzione del file system.

I journal esterni vengono creati al momento del **mkfs**, specificando i dispositivi journal al momento del montaggio. Consultate le pagine man di **mke2fs(8)**, **mkfs.xfs(8)** e **mount(8)** per maggiori informazioni.

7.1.2. Opzioni di montaggio

Barrier

Un write barrier è un meccanismo del kernel usato per assicurare la scrittura corretta e ordinata sullo storage persistente dei metadati del file system, anche quando dispositivi di storage con cache di scrittura volatile perdono alimentazione. I file system con write barrier abilitato assicurano che qualsiasi dato trasmesso tramite **fsync()**, non venga perso in presenza di un perdita di alimentazione. Red Hat Enterprise Linux abilita per impostazione predefinita un barrier su tutti gli hardware che supportano questa operazione.

Tuttavia abilitando i write barrier rallenterete in modo significativo alcune applicazioni; in particolare le applicazioni che utilizzano spesso **fsync()**, o creano e rimuovono numerosi file di piccole dimensioni. Per storage con nessun cache di scrittura volatile, o in casi rari dove è accettabile avere inconsistenze del file system e perdite di dati dopo una perdita di alimentazione, è possibile disabilitare i barrier usando l'opzione di montaggio ***nobarrier***. Per maggiori informazioni consultare la *Storage Administration Guide*.

Tempo di accesso (noatime)

Storicamente quando un file viene letto il tempo di accesso (***atime***) per quel file deve essere aggiornato nei metadati dell'inode, tale processo richiede una scrittura I/O aggiuntiva. Se non sono necessari metadati ***atime*** accurati, montate il file system con l'opzione ***noatime*** per eliminare gli aggiornamenti dei metadati. Tuttavia nella maggior parte dei casi, ***atime*** non risulta essere un problema molto grande a causa del comportamento del relative atime predefinito (o ***relatime***) nel kernel di Red Hat Enterprise Linux 6. ***relatime*** aggiorna solo ***atime*** se l'***atime*** precedente è più vecchio rispetto al tempo di modifica o "modification time" (***mtime***) o allo stato di change time (***ctime***).

NOTA

Abilitando l'opzione ***noatime*** abiliterete anche ***nodiratime***; non vi è alcun bisogno di abilitare sia ***noatime*** che ***nodiratime***.

Supporto read-ahead migliorato

Read-ahead velocizza l'accesso dei file tramite il ripristino dei dati ed il loro caricamento sulla cache della pagina, così facendo essi saranno disponibili nella memoria e non sul disco. Alcuni carichi di lavoro, come ad esempio quelli con uno streaming I/O sequenziale, traggono beneficio dai valori elevati di read-ahead.

Il tool **tuned** e l'uso della segmentazione LVM possono elevare il valore di read-ahead. Talvolta ciò non è sempre sufficiente con alcuni carichi di lavoro. Altresì Red Hat Enterprise Linux non è sempre in grado di impostare un valore read-ahead appropriato in base ai parametri rilevati del file system. Per esempio, se uno storage array molto potente si presenta a Red Hat Enterprise Linux come un LUN singolo molto potente, il sistema operativo non lo riconoscerà come un array LUN potente e, per impostazione predefinita, non utilizzerà tutti i possibili vantaggi disponibili allo storage con read-ahead.

Usare il comando **blockdev** per visualizzare e modificare il valore di read-ahead. Per visualizzare il valore corrente di read-ahead per un dispositivo a blocchi particolare eseguire:

–

```
# blockdev -getra device
```

Per modificare il valore di read-ahead per il dispositivo a blocchi in questione eseguire il seguente comando. *N* rappresenta il numero di settori a 512-byte.

```
# blockdev -setra N device
```

Da notare che il valore selezionato con il comando **blockdev** non sarà persistente dopo il riavvio. È consigliato creare uno script **init.d** del run level per impostare questo valore durante l'avvio.

7.1.3. Gestione del file system

Rimuovere i blocchi non utilizzati

Le operazioni di Batch discard e Online discard sono funzioni dei file system montati, attraverso le quali è possibile rimuovere i blocchi non usati dal file system. Queste operazioni sono utili sia per unità solid-state che per storage thinly-provisioned.

Le *operazioni di Batch discard* vengono eseguite dall'utente con il comando **fstrim**. Questo comando rimuove i blocchi non utilizzati che soddisfano i criteri di ricerca impostati dall'utente. Entrambe le operazioni sono supportate e possono essere utilizzate con file system XFS e ext4 in Red Hat Enterprise Linux 6.2 e versioni più recenti, solo se il dispositivo a blocchi sottostante al file system supporta operazioni di rimozione fisiche. Queste operazioni sono supportate se il valore di **/sys/block/*device*/queue/discard_max_bytes** non è zero.

Le *operazioni Online discard* sono operazioni specificate al momento del montaggio tramite l'opzione **-o discard** (in **/etc/fstab** o come parte del comando **mount**), ed eseguite in tempo reale senza alcun intervento dell'utente. Queste operazioni rimuovono solo blocchi in transizione con uno stato di usato ad uno disponibile. Tali operazioni sono supportate sui file system ext4 in Red Hat Enterprise Linux 6.2 e versioni più recenti e su file system XFS in Red Hat Enterprise Linux 6.4 e versioni più recenti.

Red Hat consiglia l'uso di operazioni Batch discard se il carico di lavoro del sistema lo consente. Per mantenere le prestazioni desiderate eseguire operazioni Online discard.

7.1.4. Considerazioni sull'applicazione

Pre-assegnazione

I file system ext4, XFS e GFS2 supportano una pre-assegnazione efficiente dello spazio tramite l'invocazione glibc **fallocate(2)**. In casi in cui i file possono essere segmentati incorrettamente a causa dei percorsi di scrittura, causando il deterioramento dei processi di lettura, questa tecnica può essere molto utile. Il processo di Pre-assegnazione contrassegna lo spazio del disco come se fosse stato assegnato ad un file, senza scrivere però alcun dato nello spazio in questione. Fino a quando non verranno scritti dati veri e propri all'interno del blocco pre-assegnato, le operazioni di lettura ritorneranno valori pari a zero.

7.2. PROFILI PRESTAZIONI DEL FILE SYSTEM

Il tool **tuned-adm** permette agli utenti di selezionare facilmente un certo numero di profili usati per migliorare le prestazioni in casi specifici. I profili particolarmente utili usati per il miglioramento delle prestazioni dello storage sono:

latency-performance

Un profilo server per l'ottimizzazione delle prestazioni della latenza. Disabilita i maccanismi di risparmio energetico **tuned** e **ktune**. La modalità **cpuspeed** varia in **performance**. Per ogni dispositivo l'I/O elevator viene modificato in **deadline**. Il parametro ***cpu_dma_latency*** viene registrato con un valore **0** (latenza più bassa) per la qualità di gestione dell'alimentazione, limitando la latenza quando possibile.

throughput-performance

Un profilo server per l'ottimizzazione tipica delle prestazioni di output netto. Questo profilo è consigliato se il sistema non presenta alcuno storage di classe enterprise. Simile a **latency-performance** ad eccezione:

- ***kernel.sched_min_granularity_ns*** (scheduler minimal preemption granularity) è impostato su **10** millisecondi,
- ***kernel.sched_wakeup_granularity_ns*** (scheduler wake-up granularity) impostato su **15** millisecondi,
- ***vm.dirty_ratio*** (virtual machine dirty ratio) impostato su 40%, e
- le transparent huge pages sono abilitate.

enterprise-storage

Questo profilo è consigliato per configurazioni del server con dimensioni enterprise con uno storage di classe enterprise, inclusa una protezione della cache del controller con un backup tramite batteria ed una gestione della cache su-disco. Simile al profilo **throughput-performance**, ad eccezione:

- Il valore di ***readahead*** viene impostato su **4x**, e
- file system non root/boot rimontati con **barrier=0**.

Maggiori informazioni su **tuned-adm** sono disponibili con le pagine man (**man tuned-adm**), o nella *Power Management Guide* disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

7.3. FILE SYSTEM

7.3.1. File system Ext4

Il file system ext4 è una estensione scalabile del file system ext3 predefinito disponibile in Red Hat Enterprise Linux 5. Ext4 è ora il file system predefinito per Red Hat Enterprise Linux 6 ed è supportato con una dimensione massima di 16 TB. Viene supportato altresì un file singolo con dimensione massima di 16TB. Esso rimuove il limite della sottodirectory 32000 presente in ext3.

NOTA

Per file system più grandi di 16TB è consigliato l'uso di un file system con capacità elevate scalabili come ad esempio XFS. Per maggiori informazioni consultare Sezione 7.3.2, «Il file system XFS» .

Le impostazioni predefinite del file system ext4 sono ottimali per la maggior parte dei carichi di lavoro, ma se il comportamento del file system influenza negativamente le prestazioni, allora sono disponibili diverse opzioni di ottimizzazione:

Inizializzazione tabella inode

Per file system molto grandi il processo **mkfs.ext4** può richiedere un periodo di tempo molto lungo durante l'inizializzazione di tutte le tabelle inode nel file system. Questo processo può essere rinviato con l'opzione ***-E lazy_itable_init=1***. Se utilizzate questa impostazione i processi del kernel continueranno ad inizializzare il file system dopo il suo processo di montaggio. La velocità con la quale viene eseguito questo processo può essere controllata con l'opzione ***-o init_itable=n*** del comando **mount**, dove la quantità di tempo per eseguire il processo è di circa 1/n. Il valore predefinito per **n** è **10**.

Comportamento di Auto-fsync

Poichè alcune applicazioni non eseguono un **fsync()** corretto dopo la modifica del nome di un file esistente, o dopo un processo di riscrittura o troncamento, ext4 esegue il default su sincronizzazione automatica dei file dopo operazioni replace-via-rename e replace-via-truncate. Questo comportamento è consistente con i comportamenti del file system ext3 più vecchi. Tuttavia le operazioni **fsync()** possono richiedere molto tempo, per questo motivo se il suddetto processo automatico non è necessario usare l'opzione ***-o noauto_da_alloc*** con il comando **mount** per disabilitarlo. Ciò significa che l'applicazione deve esplicitamente usare **fsync()** per rendere i dati persistenti.

Priorità I/O del journal

Per impostazione predefinita viene conferita al journal commit I/O una priorità leggermente più elevata rispetto all'I/O normale. Tale priorità può essere controllata con l'opzione ***journal_ioprio=n*** del comando **mount**. Il valore predefinito è **3**. La gamma di valori validi va da 0 a 7, il valore 0 rappresenta la priorità più alta di I/O.

Per altre opzioni **mkfs** e di ottimizzazione consultare le pagine man di **mkfs.ext4(8)** e **mount(8)** ed il file **Documentation/filesystems/ext4.txt** nel pacchetto kernel-doc.

7.3.2. Il file system XFS

XFS è un file system con journal robusto e altamente scalabile a singolo host a 64-bit. Esso è interamente basato sull'estensione e quindi supporta dimensioni di file system e di file molto grandi. Il numero di file che un sistema XFS può contenere è limitato solo dallo spazio disponibile nel file system.

XFS supporta il journaling dei metadati e facilita il ripristino più veloce da un crash. Il file system XFS può essere deframmentato ed esteso anche se montato ed attivo. Altresì Red Hat Enterprise Linux 6 supporta il backup ed il ripristino delle utilità specifiche a XFS.

XFS utilizza una assegnazione basata sulle estensioni e presenta un certo numero di schemi di assegnazione come ad esempio quella ritardata e la pre-assegnazione esplicita. L'assegnazione basata sulle estensioni "extent-based" fornisce un metodo più compatto ed efficiente di monitoraggio dello spazio usato in un file system, e migliora le prestazioni di file molto grandi riducendo la frammentazione e lo spazio usato dai metadati. L'assegnazione ritardata "delayed allocation" aumenta le possibilità che un file venga scritto in un gruppo adiacente di blocchi, riducendone la frammentazione e migliorando le prestazioni. È possibile usare una pre-assegnazione per impedire qualsiasi processo di frammentazione in casi dove l'applicazione è a conoscenza della quantità di dati da scrivere.

XFS fornisce una scalabiltà I/O eccellente grazie all'uso di b-trees per indicizzare tutti i metadati ed i dati dell'utente. I conteggi degli oggetti aumenta poichè tutte le operazioni presenti negli indici

ereditano le caratteristiche di scalabilità logaritmica del b-trees sottostante. Alcune delle opzioni di ottimizzazione fornite da XFS al momento del **mkfs**, variano l'ampiezza di b-trees, modificando così le caratteristiche di scalabilità dei diversi sottosistemi.

7.3.2.1. Ottimizzazione di base di XFS

In generale le opzioni di montaggio ed il formato XFS predefinito sono ottimali per la maggior parte dei carichi di lavoro; Red Hat consiglia di usare i valori predefiniti se non siete in presenza di modifiche alla configurazione in grado di alterare i requisiti specifici del carico di lavoro. **mkfs.xfs** esegue automaticamente una configurazione con l'ampiezza e l'unità del segmento corretto per uniformarsi all'hardware. Sarà necessario eseguire una configurazione manuale se si utilizza un hardware RAID.

L'opzione di montaggio **inode64** è fortemente consigliata per file system multi-terabyte; tale opzione non è consigliata se il file system viene esportato tramite NFS ed i client NFS a 32-bit precedenti hanno bisogno di un accesso al file system.

È consigliato usare l'opzione di montaggio **logbsize** con file system frequentemente modificati o in bursts. Il valore predefinito è **MAX** (unità segmento di registrazione a 32 KB), con dimensione massima di 256 KB. Un valore di 256 KB è consigliato per file system che hanno subito numerose modifiche.

7.3.2.2. Ottimizzazione avanzata per XFS

Prima di modificare i parametri di XFS è necessario capire perchè i suoi parametri predefiniti possono causare problemi alle prestazioni. Per fare questo è necessario comprendere il comportamento dell'applicazione interessata e le reazioni del file system alle suddette operazioni.

Le problematiche relative alle prestazioni che si possono correggere o ridurre tramite un processo di ottimizzazione, vengono generalmente causate da una frammentazione del file o da un conflitto di risorse nel file system. Per risolvere questi problemi sono disponibili diversi metodi, e in alcuni casi la correzione del problema richiederà la modifica dell'applicazione e non della configurazione del file system.

Se non avete eseguito prima questo processo è consigliato consultare il Red Hat support engineer.

Ottimizzazione di numerosi file

XFS impone un limite arbitrario sul numero di file archiviabili su un file system. In generale questo limite è sufficientemente elevato da non essere mai superato. Se credete che il limite predefinito non sia sufficiente, aumentate la percentuale di spazio del file system per gli inode tramite il comando **mkfs.xfs**. Se raggiungete il limite del file dopo la creazione del file system (generalmente indicato da errori ENOSPC durante il tentativo di creare un file o directory anche se lo spazio è disponibile), è possibile modificare il limite con il comando **xfs_growfs**.

Ottimizzazione di numerosi file in una directory

Per l'intero ciclo di vita di un file system la dimensione di un blocco della directory è fissa e può essere modificata solo durante la formattazione iniziale con il comando **mkfs**. La dimensione minima del blocco della directory corrisponde a quella del blocco del file system, che per impostazione predefinita è su **MAX** (4 KB). In generale non vi è alcun motivo per ridurre la dimensione del blocco della directory.

Poichè le struttura della directory si basa su b-tree, la modifica della dimensione del blocco può cambiare la quantità delle informazioni della directory ripristinabili o modificabili per I/O fisico. Più grande è la directory, maggiore è la quantità di I/O necessaria ad una operazione per la dimensione di un dato blocco.

Tuttavia se utilizzate dimensioni del blocco della directory più grandi, ogni operazione di modifica può utilizzare più CPU rispetto alla stessa operazione su un file system con una dimensione del blocco più

piccola. Ciò significa che per directory più piccole, dimensioni del blocco più grandi possono ridurre le prestazioni del processo di modifica. Quando una directory raggiunge una dimensione dove l'I/O risulta essere il fattore che limitale prestazioni, le directory con un blocco più grande hanno migliori prestazioni.

La configurazione predefinita che comprende un file system con blocco di dimensioni di 4 KB e una directory con un blocco di 4 KB, è quella più idonea per directory con un numero di voci di 1-2 milioni e con lunghezza del nome pari a 20-40 byte per voce. Se un file system ha bisogno di un numero maggiore di voci, le directory con un blocco più grande presentano migliori prestazioni - una dimensione del blocco di 16 KB è più idoneo per file system con un numero di voci pari a 1-10 milioni, mentre un blocco di 64 KB è il più idoneo per file system con un numero di voci maggiore a 10 milioni.

Se il carico di lavoro utilizza le ricerche randomiche della directory più delle modifiche (cioè le letture delle directory risultano essere più comuni o importanti rispetto ai processi di scrittura), i limiti sopra indicati per l'aumento della dimensione del blocco saranno approssimativamente inferiori.

Ottimizzazione tramite processi simultanei

Diversamente da altri file system, XFS è in grado di assegnare e deallocare simultaneamente se tali operazioni si verificano su oggetti non condivisi. È possibile eseguire simultaneamente le suddette operazioni nei confronti delle estensioni se i gruppi risultano essere diversi. In modo simile i processi di assegnazione/deallocazione di inode si possono verificare se tali operazioni interessano gruppi diversi.

Il numero dei gruppi di assegnazione è importante quando si utilizzano macchine con un conteggio CPU elevato e applicazioni multi-threaded che tentano di eseguire operazioni simultanee. Se esistono solo quattro gruppi di assegnazione, operazioni parallele e continuate di metadati interesseranno solo le quattro CPU (il limite fornito dal sistema). Per file system piccoli, assicuratevi che il numero di gruppi di assegnazione sia supportato dal limite di operazioni simultanee fornito dal sistema. Per file system grandi (decine di terabyte o maggiori) le opzioni di formattazione predefinite generalmente sono in grado di creare un numero di gruppi sufficiente per non limitare questo processo.

Le applicazioni devono essere a conoscenza dei punti di contesa per poter eseguire operazioni in parallelo, capacità ereditata nella struttura del file system XFS. Non è possibile modificare simultaneamente una directory, per questo motivo le applicazioni in grado di creare e rimuovere un numero molto grande di file, non dovrebbero eseguire l'archiviazione in una singola directory. Ogni directory creata viene posizionata in un gruppo di assegnazione diverso, quindi tecniche come il file hashing su directory multiple forniscono un percorso di archiviazione più scalabile rispetto all'uso di una directory singola molto grande.

Ottimizzazione delle applicazioni che utilizzano attributi estesi

Se lo spazio è disponibile sull'inode, XFS è in grado di archiviare attributi direttamente al suo interno. Se l'attributo è idoneo per l'inode, esso potrà essere ripristinato e modificato senza alcun I/O aggiuntivo per il recupero di blocchi separati dell'attributo. Le prestazioni per attributi out-of-line (attributi non presenti all'interno dell'inode in questione) saranno più basse rispetto a quelle degli attributi in-line (presenti all'interno dell'inode in questione).

Per la dimensione predefinita dell'inode pari a 256 byte, 100 byte di spazio circa è disponibile in base al numero di puntatori dell'estensione dei dati archiviati nell'inode. La dimensione predefinita dell'inode è utile per l'archiviazione di un numero ridotto di piccoli attributi.

Aumentando la dimensione dell'inode al momento di mkfs è possibile aumentare lo spazio disponibile per l'archiviazione degli attributi "in-line". Un inode di 512 byte aumenta lo spazio disponibile per gli attributi a circa 350 byte, un inode di 2 KB avrà circa 1900 byte di spazio disponibile.

Tuttavia è presente un limite sulla dimensione dei singoli attributi archiviabili in-line - infatti è presente un limite massimo di 254 byte sia per l'attributo name che per value (attributi name e value di 254 byte verranno classificati come in-line). Se questi limiti vengono superati, gli attributi verranno

forzati fuori (out-of-line) anche se è disponibile spazio sufficiente per archiviare tutti gli attributi nell'inode.

Ottimizzazione per modifiche sostenute dei metadati

La dimensione del log è il fattore principale per determinare il livello di modifica sostenuto dai metadati. Il dispositivo di log è circolare, quindi prima di poter sovrascrivere la coda tutte le modifiche presenti nel log devono essere salvate nelle posizioni reali sul disco. Tale operazione può richiedere tentativi ripetuti di riscrittura di metadati corrotti. La configurazione predefinita modifica la dimensione del log in base alla dimensione generale del file system, quindi in molti casi la dimensione non avrà bisogno di alcuna ottimizzazione.

Se implementate un dispositivo di log piccolo si verificheranno processi di writeback frequenti dei metadati - il log avrà la necessità di cancellare le voci più vecchie per liberare spazio, così facendo i metadati modificati verranno continuamente scritti sul disco causando una riduzione nelle prestazioni delle operazioni.

Aumentando la dimensione del log verrà aumentato il periodo di tempo tra gli eventi di tail pushing. Ciò permette una migliore aggregazione dei metadati "sporchi" e migliori pattern di writeback dei metadati insieme ad un numero minore di writeback dei metadati modificati più frequentemente. Uno degli aspetti negativi è rappresentato dai log più grandi i quali richiedono una memoria maggiore per monitorare tutte le modifiche ancora presenti della memoria.

Se siete in possesso di una macchina con una memoria limitata i log più grandi non rappresentano un fattore positivo, questo poichè le limitazioni della memoria causeranno un writeback dei metadati, rimuovendo così tutti i benefici dovuti alla presenza di un log molto grande. In questi casi, spesso i log più piccoli possono fornire una migliore prestazione poichè il writeback dei metadati del log con uno spazio ridotto è molto più efficiente rispetto al writeback eseguito in base al memory reclamation.

È sempre consigliato cercare di allineare il log con il segmento sottostante del dispositivo che contiene il file system. Su dispositivi MD e DM, **`mkfs`** esegue questa operazione per impostazione predefinita, ma per hardware RAID sarà necessario specificarla. Una impostazione corretta impedirà ad un log I/O di generare un I/O non allineato, quindi operazioni read-modify-write al momento della scrittura delle modifiche sul disco.

Le operazioni di log possono essere ulteriormente migliorate modificando le opzioni di montaggio. Aumentando la dimensione dei buffer log in-memoria (**`logbsize`**), aumenterete la velocità con la quale le modifiche verranno scritte sul log. La dimensione del buffer log predefinita è **`MAX`** (32 KB, unità segmento di log), con dimensione massima di 256 KB. In generale, un valore più grande corrisponderà a prestazioni più veloci. Tuttavia con carichi di lavoro fsync molto grandi, buffer log piccoli possono essere più veloci rispetto a buffer più grandi con un allineamento maggiore dell'unità del segmento.

Anche l'opzione di montaggio **`delaylog`** migliora le prestazioni per le modifiche sostenute dei metadati tramite la riduzione del numero di modifiche per il log. Per fare questo le modifiche vengono raggruppate nella memoria prima di essere salvate sul log: i metadati modificati con molta frequenza vengono scritti periodicamente sul log e non ad ogni verifica. Questa opzione migliora l'uso della memoria durante il monitoraggio dei metadati corrotti, ed aumenta il numero di possibili operazioni perse in presenza di un crash, ma al tempo stesso migliora la velocità di modifica dei metadati e la scalabilità. L'uso di questa opzione non riduce l'integrità dei dati o metadati quando si utilizzano **`fsync`**, **`fdatasync`** o **`sync`** per la loro scrittura sul disco.

7.4. CLUSTERING

Uno storage clusterizzato fornisce una immagine uniforme del file system su tutti i server presenti in un cluster, permettendo così ai server di eseguire un processo di lettura e scrittura su un singolo file system condiviso. Tale impostazione semplifica l'amministrazione dello storage limitando compiti di

tipo installazione e patching delle applicazione su un file system. Un file system per l'intero cluster elimina la necessità di usare copie ridondanti di dati dell'applicazione, semplificando così i processi di backup e di disaster recovery.

L'High Availability Add-On di Red Hat fornisce uno storage clusterizzato insieme al Red Hat Global File System 2 (parte del Resilient Storage Add-On).

7.4.1. Global File System 2

Il Global File System 2 (GFS2) è un file system nativo che interfaccia direttamente l'interfaccia del file system del kernel di Linux. Esso permette a computer multipli (nodi) di condividere simultaneamente lo stesso dispositivo di storage in un cluster. Il file system GFS2 esegue regolazioni automatizzate e permette al tempo stesso di eseguire simili operazioni manualmente. Questa sezione contiene alcune considerazioni su come eseguire una ottimizzazione manuale.

Red Hat Enterprise Linux 6.4 introduce alcuni miglioramenti sulla gestione della frammentazione dei file con GFS2. I file creati da Red Hat Enterprise Linux 6.3 o versioni meno recenti, erano propensi alla frammentazione se file multipli venivano archiviati contemporaneamente da più di un processo. Tale frammentazione rallentava l'esecuzione generale in modo particolare con carichi di lavoro in presenza di file molto grandi. Con Red Hat Enterprise Linux 6.4 i processi di scrittura simultanei risultano in una frammentazione del file minore, garantendo così migliori prestazioni in presenza di carichi di lavoro di questo tipo.

Anche se non è presente alcuno strumento di deframmentazione per GFS2 su Red Hat Enterprise Linux, è possibile eseguire una deframmentazione di file singoli identificandoli con **filefrag**, copiandoli su file temporanei e modificandone il nome per sostituire quelli originali. (È possibile eseguire questa operazione in versioni precedenti alla 6.4 se il processo di scrittura viene eseguito in modo sequenziale.)

Poichè il GFS2 utilizza un meccanismo di bloccaggio globale il quale può richiedere una comunicazione tra i nodi di un cluster, la prestazione migliore avviene quando il sistema viene creato per impedire eventuali contese dei nodi in un file o directory. Di seguito vengono riportati alcuni di questi metodi:

- Preassegnare file e directory con **`fallocate`** quando possibile, per ottimizzare il processo di assegnazione ed evitare la necessità di bloccare le pagine d'origine.

- MInimizzare le aree del file system condivise tra nodi multipli per ridurre i rischi di invalidazione cross-node cache migliorando le prestazioni. Per esempio, se numerosi nodi montano lo stesso file system ma accedono a directory secondarie diverse, probabilmente avrete migliori prestazioni spostando una sottodirectory su un file system separato.

- Selezionare un numero ed una dimensione ottimali del gruppo di risorse. Ciò dipende dalle dimensioni del file tipiche e dallo spazio disponibile sul sistema e può interessare la possibilità che nodi multipli cercheranno di usare simultaneamente un gruppo di risorse. Troppi gruppi possono rallentare l'assegnazione durante la ricerca dello spazio, mentre un numero troppo ristretto di gruppi possono causare una contesa del blocco durante l'annullamento dell'assegnazione. È consigliato provare configurazioni multiple per determinare il processo ottimale per il carico di lavoro desiderato.

Tuttavia la contesa non rappresenta il solo problema in grado di interessare le prestazioni del file system GFS2. Altre pratiche per migliorare le prestazioni generali sono:

- Selezionare l'hardware di storage in base ai percorsi I/O previsti dei nodi del cluster e ai requisiti delle prestazioni del file system.

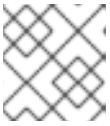
- Usare uno storage "solid-state" quando possibile per ridurre il tempo di ricerca "seek time".

- Creare un file system con dimensioni appropriate per il carico di lavoro desiderato ed assicurarsi che esso non raggiunga mai una capacità di 80%. File system più piccoli avranno tempi di backup proporzionalmente più piccoli, e avranno bisogno di un periodo e di una memoria minori per l'esecuzione dei controlli del file system. Essi soggetti ad una elevata frammentazione se troppo piccoli per il carico di lavoro desiderato.

- Impostare dimensioni più grandi per il journal con carichi di lavoro intensi per i metadati o durante l'utilizzo dei dati presenti nel journal. Tale processo richiede una quantità di memoria maggiore, ma garantisce elevate prestazioni poichè dispone di spazio libero nel journal per l'archiviazione dei dati prima di dover eseguire un processo di scrittura.

- Assicuratevi che gli orologi presenti nei nodi del GFS2 siano sincronizzati per evitare possibili problematiche con applicazioni di rete. È consigliato usare un NTP (Network Time Protocol).

- Se i tempi di accesso alle directory o ai file non sono critici per il normale svolgimento delle operazioni dell'applicazione, montare il file system con le opzioni di montaggio **`noatime`** e **`nodiratime`**.



NOTA

Red Hat consiglia l'uso dell'opzione **`noatime`** con GFS2.

- Se desiderate usare un quota, provate a ridurre la frequenza del numero di operazioni di sincronizzazione, oppure usate una sincronizzazione fuzzy quota per impedire eventuali problemi di prestazioni causati dai continui aggiornamenti del file quota.

NOTA

Il fuzzy quota accounting permette a utenti o gruppi di superare di poco i limiti impostati. Per minimizzare questa tendenza, GFS2 riduce dinamicamente il periodo di sincronizzazione quando si è prossimi ad un limite quota "hard".

Per maggiori informazioni su ogni aspetto dell'ottimizzazione delle prestazioni di GFS2 consultare la *Global File System 2*guide disponibile su
http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

CAPITOLO 8. NETWORKING

Col tempo lo stack di rete di Red Hat Enterprise Linux è stato aggiornato con numerose funzioni di ottimizzazione automatizzate. Per la maggior parte dei carichi di lavoro le impostazioni di rete auto-configurate forniscono una prestazione ottimale.

In molti casi le problematiche relative alle prestazioni sono causate da un funzionamento non corretto dell'hardware o per un errore dell'infrastruttura. Questi casi non vengono affrontati nel documento poichè le soluzioni e le problematiche discusse in questo capitolo, sono utili per l'ottimizzazione di sistemi con un funzionamento corretto.

Il Networking è un sistema molto delicato che contiene diverse sezioni con connessioni molto sensibili. Ecco perchè la community della open source e Red Hat investono gran parte del loro lavoro per ottimizzare automaticamente le prestazioni della rete. Per questo motivo dati i carichi di lavoro potreste non aver bisogno di riconfigurare il networking per migliorare le prestazioni.

8.1. MIGLIORAMENTI DELLE PRESTAZIONI DI RETE

Red Hat Enterprise Linux 6.1 apporta i seguenti miglioramenti delle prestazioni di rete:

Receive Packet Steering (RPS)

RPS abilita una coda **rx** del NIC singola in modo da avere il proprio carico di lavoro **softirq** distribuito tra diverse CPU. Questa impostazione aiuta a prevenire la congestione del traffico di rete sulla coda dell'hardware NIC singola.

Per abilitare RPS specificare i nomi di destinazione delle CPU in **/sys/class/net/*ethX*/queues/*rx-N*/rps_cpus**, sostituendo ***ethX*** con il nome del dispositivo corrispondente al NIC (per esempio **eth1**, **eth2**) e ***rx-N*** con la coda di ricezione del NIC specificato. Ciò permetterà alle CPU specificate nel file di processare i dati dalla coda ***rx-N*** su ***ethX***. Quando specificate le CPU prendete in considerazione il *cache affinity* della coda [4].

Receive Flow Steering

RFS è una estensione di RPS e permette all'amministratore di configurare una tabella hash popolata automaticamente quando le applicazioni ricevono i dati e vengono interrogate dallo stack di rete. Così facendo verranno determinate le applicazioni che ricevono ogni sezione di dati della rete (in base alle informazioni della rete source:destination).

Utilizzando queste informazioni lo stack di rete sarà in grado di programmare la CPU ottimale per la ricezione di ogni pacchetto. Per configurare RFS usare i seguenti parametri:

/proc/sys/net/core/rps_sock_flow_entries

Usato per controllare il numero massimo di socket/flussi che il kernel è in grado di dirigere verso la CPU specificata. Questo rappresenta un limite condiviso da tutto il sistema.

/sys/class/net/*ethX*/queues/*rx-N*/rps_flow_cnt

Usato per controllare il numero massimo di socket/flussi che il kernel è in grado di dirigere verso la coda di ricezione specificata (***rx-N***) su di un NIC (***ethX***). Da notare che la somma di tutti i valori per-coda per questo parametro su tutti i NIC, deve essere uguale o minore a quello di **/proc/sys/net/core/rps_sock_flow_entries**.

Diversamente da RPS, RFS permette sia alla coda di ricezione che all'applicazione di condividere la stessa CPU durante la processazione del flusso di pacchetti. Questa impostazione in alcuni casi migliora le prestazioni. Tuttavia questi miglioramenti dipendono da fattori tipo gerarchia della cache, carico dell'applicazione ecc.

Supporto getsockopt per TCP thin-stream

Thin-stream è un termine usato per indicare i protocolli di trasporto dove le applicazioni inviano dati ad una velocità così bassa che i meccanismi di ritrasmissione del protocollo non risultano completamente saturi. Le applicazioni che utilizzano protocolli thin-stream eseguono un trasporto usando protocolli affidabili come TCP; in molti casi queste applicazioni forniscono servizi di tipo time-sensitive (per esempio stock trading, giochi online, sistemi di controllo).

Per servizi "time-sensitive" la perdita dei pacchetti può essere devastante per la qualità del servizio. Per prevenirla è stata migliorata la chiamata **`getsockopt`** che ora supporta due opzioni aggiuntive:

TCP_THIN_DUPACK

Per thin stream, questo valore booleano abilita l'attivazione dinamica delle ritrasmissioni dopo un dupACK.

TCP_THIN_LINEAR_TIMEOUTS

Per thin stream, questo valore booleano abilita l'attivazione dinamica dei timeout lineari.

Entrambe le opzioni sono attivate dall'applicazione. Per maggiori informazioni consultare **`file:///usr/share/doc/kernel-doc-`*`version`*`/Documentation/networking/ip-sysctl.txt`**. Per maggiori informazioni sui thin-stream consultare **`file:///usr/share/doc/kernel-doc-`*`version`*`/Documentation/networking/tcp-thin.txt`**.

Supporto Transparent Proxy (TProxy)

Il kernel supporta ora i socket UDP e IPv4 TCP assegnati non-localmente per il supporto del transparent proxy. Per poterlo abilitare configurare iptables in modo adeguato. Sarà necessario altresì abilitare e configurare correttamente la politica di instradamento.

Per maggiori informazioni sui transparent proxy consultare **`file:///usr/share/doc/kernel-doc-`*`version`*`/Documentation/networking/tproxy.txt`**.

8.2. IMPOSTAZIONI DI RETE OTTIMIZZATE

Il processo di ottimizzazione delle prestazioni è un processo eseguito a priori. Spesso vengono modificate variabili conosciute, prima di eseguire un'applicazione o implementare un sistema. Se la modifica è inefficace, allora si passa alla modifica di altre variabili. Questa logica *deriva dal fatto che* per l'utente il sistema non opera sempre su livelli ottimali; Per questo motivo generalmente si *apportano* le modifiche ritenute necessarie. In alcuni casi esguiamo queste modifiche seguendo delle ipotesi calcolate.

Come precedentemente indicato lo stack di rete esegue processi di ottimizzazione automatizzati. È necessario quindi capire il processo di ottimizzazione della rete, ed in particolare del funzionamento dello stack di rete e dei requisiti delle risorse di rete del sistema specifico. Configurazioni di rete incorrette possono ridurre sensibilmente le prestazioni.

Per esempio, considerate il problema del *bufferfloat*. L'aumento della profondità della coda del buffer

causa alle connessioni TCP una congestione delle finestre maggiore rispetto a quella prevista dal link (a causa della profondità del buffer). Tuttavia queste connessioni presentano altresì valori RRT molto grandi, ciò è dovuto alla quantità di tempo che i frame trascorrono in-coda. Tale situazione causa un output non ottimale poichè diviene impossibile rilevare la presenza di una congestione.

Per quanto riguarda le prestazioni della rete è consigliato mantenere le impostazioni predefinite, *a meno che* un particolare problema diventi apparente. Questi problemi includono una perdita dei frame, output netto ridotto e altri problemi simili. Anche in tali situazioni è consigliato verificare in modo dettagliato il tipo di errore e non semplicemente modificare le impostazioni (ad esempio aumentare la lunghezza della coda/buffer, ridurre la latenza delle interruzioni, ecc).

Per diagnosticare in modo corretto la presenza di un problema utilizzare il seguente strumento:

netstat

Una utilità a linea di comando che stampa le connessioni di rete, tabelle di instradamento, statistiche dell'interfaccia, connessioni di mascheramento e appartenenze multicast. Essa è in grado di ripristinare le informazioni del sottosistema di rete dal file system **`/proc/net/`**. I file includono:

- **`/proc/net/dev`** (informazioni del dispositivo)
- **`/proc/net/tcp`** (Informazioni del socket TCP)
- **`/proc/net/unix`** (Informazioni socket del dominio di Unix)

Per maggiori informazioni su **`netstat`** e sui file di riferimento di **`/proc/net/`** consultare la pagina man di **`netstat`**: **`man netstat`**.

dropwatch

Una utilità di monitoraggio che controlla la perdita dei pacchetti da parte del kernel. Per maggiori informazioni consultare la pagina man di **`dropwatch`**: **`man dropwatch`**.

ip

Una utilità per la gestione ed il controllo degli instradamenti, dispositivi, politica di instradamento e tunnel. Per maggiori informazioni consultare la pagina man di **`ip`**: **`man ip`**.

ethtool

Utilità per la visualizzazione e la modifica delle impostazioni del NIC. Per maggiori informazioni consultare la pagina man di **`ethtool`**: **`man ethtool`**.

/proc/net/snmp

Un file per la visualizzazione dei dati ASCII necessari per le informazioni sulla gestione dell'IP, ICMP, TCP e UDP per un **`snmp`** agent. Usato anche per la visualizzazione di statistiche UDP-lite in tempo reale.

La *SystemTap Beginners Guide* contiene numerosi script d'esempio da usare per registrare e controllare le prestazioni della rete. Questa guida è disponibile su http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

Dopo aver raccolto i dati utili sui problemi presenti nelle prestazioni della rete, sarà possibile formulare una teoria — e probabilmente avere una soluzione. [5] Per esempio, un aumento degli errori di input UDP in **`/proc/net/snmp`** indica che una o più code di ricezione del socket risultano essere

sature quando lo stack di rete tenta di mettere in coda nuovi frame in un socket dell'applicazione.

Ciò indica la presenza di alcuni problemi per i pacchetti in *almeno* una coda. In questo questa situazione potrebbe indicare che la coda del socket passa pacchetti con una velocità molto bassa, o che il volume dei pacchetti è troppo grande per la coda. In presenza di quest'ultimo scenario è consigliato verificare i log di qualsiasi applicazione che utilizza una quantità di rete molto elevata e controllare la presenza di dati persi -- per risolvere questo problema ottimizzare o riconfigurare l'applicazione interessata.

Dimensione del buffer di ricezione del socket

Le domensioni per la ricezione e l'invio dei Socket sono modificate dinamicamente, per questo motivo non vengono quasi mai modificati manualmente. Se analisi aggiuntive come quelle presenti nell'esempio della rete di SystemTap, **`sk_stream_wait_memory.stp`**, indicano che la velocità per la gestione di una coda del socket è troppo bassa, aumentate la profondità della coda del socket dell'applicazione. Per fare questo aumentate la dimensione dei buffer di ricezione usati dal socket, configurando uno dei seguenti valori:

rmem_default

Un parametro del kernel che controlla la dimensione *predefinita* dei buffer di ricezione usati dai socket. Per la configurazione eseguire il seguente comando:

```
sysctl -w net.core.rmem_default=N
```

Sostituire ***N*** con la dimensione del buffer desiderata, in byte. Per determinare il valore di questo parametro del kernel consultare **`/proc/sys/net/core/rmem_default`**. Ricordate che il valore di **`rmem_default`** non dovrebbe essere maggiore di **`rmem_max`** (**`/proc/sys/net/core/rmem_max`**); se necessario aumentare il valore di **`rmem_max`**.

SO_RCVBUF

Una opzione del socket in grado di controllare la dimensione *massima* di un buffer di ricezione del socket, in byte. Per maggiori informazioni su **SO_RCVBUF** consultare la pagina man: **`man 7 socket`**.

Per configurare **SO_RCVBUF** usare l'utilità **`setsockopt`**. È possibile ottenere il valore corrente di **SO_RCVBUF** usando il comando **`getsockopt`**. Per maggiori informazioni sull'utilizzo delle utilità consultare la pagina man di **`setsockopt`**: **`man setsockopt`**.

8.3. PANORAMICA SULLA RICEZIONE DEI PACCHETTI

Per analizzare nel modo migliore le problematiche relative alle prestazioni della rete, è necessario comprendere il funzionamendo della ricezione dei pacchetti. Il processo di ricezione è molto importante per l'ottimizzazione delle prestazioni di una rete, poichè spesso i frame vengono persi nel percorso di ricezione. Tale perdita può causare seri problemi di prestazione per una rete.

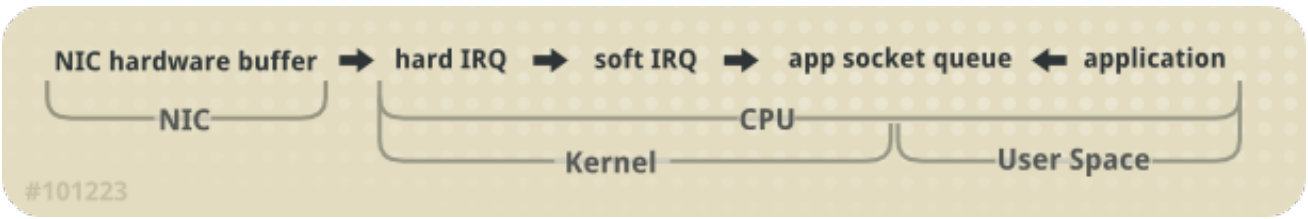


Figura 8.1. Diagramma del percorso di ricezione della rete

Il kernel di Linux riceve un frame ed esegue un processo di quattro fasi:

1. *Ricezione Hardware*: il *network interface card* (NIC) riceve il frame sul cavo. In base alla configurazione del driver il NIC trasferisce il frame ad una memoria del buffer hardware interna o ad un buffer circolare specificato.

2. *Hard IRQ*: il NIC indica la presenza di un frame della rete interrompendo la CPU. Questo processo causa la ricezione da parte dell'unità NIC di questa interruzione e programma a sua volta una *operazione soft IRQ*.

3. *Soft IRQ*: questa fase implementa il processo vero e proprio di ricezione del frame, eseguendolo in un contesto **`softirq`**. Ciò sognifica che questa fase anticipa tutte le applicazioni in esecuzione sulla CPU specificata, permettendo di avere processi hard IRQ

   In questo contesto (esecuzione sulla stessa CPU come hard IRQ, minimizzando un sovraccarico di blocco), il kernel rimuove il frame dai NIC Hardware Buffer e lo processa attraverso lo stack di rete. Da qui il frame viene inoltrato, scartato o passato ad un socket in ascolto di destinazione.

   Se passato ad un socket, il frame viene associato ad una applicazione che detiene il socket stesso. Questo processo viene eseguito in modo interattivo fino a quando il NIC Hardware Buffer non possiede alcun frame, o fino al raggiungimento del *device weight* (**`dev_weight`**). Per maggiori informazioni sul device weight consultare Sezione 8.4.1, «NIC Hardware Buffer»

4. *Ricezione applicazione*: l'applicazione riceve il frame e lo rimuove dalla coda di qualsiasi socket posseduto tramite le chiamate POSIX standard (**`read`**, **`recv`**, **`recvfrom`**). A questo punto i dati ricevuti tramite la rete non saranno più presenti nello stack.

Affinità CPU/cache

Per un output netto elevato sul percorso di ricezione è consigliato mantenere la cache L2 *attiva*. Come precedentemente indicato, i buffer di rete vengono ricevuti sulla stessa CPU dell'IRQ che ha indicato la loro presenza. Ciò significa che i dati del buffer saranno presenti sulla cache L2 della CPU di ricezione.

Per trarre vantaggio da questa impostazione, implementate una affinità del processo su applicazioni che dovranno ricevere la quantità maggiore di dati sul NIC che condivide lo stesso core della cache L2. Così facendo aumenterete le possibilità di una cache hit e quindi di migliorare sensibilmente le prestazioni.

8.4. RISOLUZIONE PROBLEMATICHE COMUNI DI PERDITA DEI FRAME/MESSA IN CODA

In numerosi casi la perdita dei frame è dovuta alla *saturazione della coda* Il kernel imposta un limite di lunghezza della coda la quale in alcuni casi si può saturare molto velocemente, se questa situazione si protrae per un periodo di tempo molto lungo si può verificare una perdita dei frame.

Come illustrato in Figura 8.1, «Diagramma del percorso di ricezione della rete» sono presenti due code principali nel percorso di ricezione: il NIC Hardware Buffer e la coda del socket. Entrambe queste code devono essere configurate in modo corretto per impedirne la loro saturazione.

8.4.1. NIC Hardware Buffer

Il NIC riempie il proprio buffer hardware con numerosi frame; il buffer viene svuotato da **`softirq`** il quale viene invocato dal NIC tramite una interruzione. Per interrogare lo stato della coda usare il seguente comando:

```
ethtool -S ethX
```

Sostituire ***ethX*** con il nome del dispositivo corrispondente del NIC. Ciò mostrerà il numero di frame persi all'interno di ***ethX***. Spesso le perdite si verificano a causa di una carenza di spazio del buffer usato per archiviare i frame.

Per risolvere questo problema usare i metodi di seguito riportati:

Traffico

Per impedire la saturazione della coda rallentare il traffico inserito. Per fare questo filtrare il numero di gruppi multicast, riducendo il traffico di trasmissione e operazioni simili.

Lunghezza della coda

Alternativamente aumentate la lunghezza della coda. Per fare questo è necessario aumentare il numero di buffer in una coda specifica fino a raggiungere il livello massimo permesso dal driver. Per fare questo modificate i parametri **rx/tx** di ***ethX*** usando:

```
ethtool --set-ring ethX
```

Inserire i valori **rx** o **tx** al comando sopra indicato. Per maggiori informazioni consultare **man ethtool**.

Peso del dispositivo

È possibile aumentare la velocità con la quale viene "svuotata" una coda. Per fare questo modificare il *peso del dispositivo* del NIC in modo appropriato. Questo attributo si riferisce al numero massimo di frame che il NIC è in grado di ricevere prima che il contesto **softirq** rilasci la CPU modificando la sua programmazione. L'attributo è controllato da **/proc/sys/net/core/dev_weight**.

Molti amministratori tendono ad usare la terza opzione. Tuttavia ricordate che selezionando questa opzione sono presenti alcune limitazioni. Aumentando il numero dei frame ricevibili da un NIC in una iterazione, si avranno cicli aggiuntivi della CPU durante i quali non sarà possibile programmare alcuna applicazione sulla CPU interessata.

8.4.2. Coda del socket

In modo simile alla coda dell'hardware NIC la coda del socket viene riempita dallo stack di rete di **softirq**. Successivamente le applicazioni possono svuotare la coda dai socket corrispondenti tramite **read**, **recvfrom** o entità simili.

Per controllare lo stato di questa coda usare l'utilità **netstat**; la colonna **Recv-Q** mostra la dimensione della coda. In generale se si verifica una saturazione essa verrà gestita in modo simile a quelle del NIC hardware buffer (es. Sezione 8.4.1, «NIC Hardware Buffer»):

Traffico

Usate la prima opzione per rallentare il traffico degli input configurando la velocità con la quale viene riempita la coda. Per fare questo filtrare o rimuovere i frame. È possibile altresì rallentare il traffico riducendo il peso del dispositivo del NIC[6].

Profondità coda

È possibile impedire la saturazione della coda aumentandone la profondità. Per fare questo aumentare il valore del parametro del kernel **rmem_default** o dell'opzione **SO_RCVBUF**. Per maggiori informazioni consultare Sezione 8.2, «Impostazioni di rete ottimizzate» .

Frequenza chiamata dell'applicazione

Quando possibile ottimizzare l'applicazione in modo da aumentare la frequenza delle chiamate. Per fare questo modificare o riconfigurare l'applicazione della rete per eseguire un numero maggiore di chiamate POSIX (come ad esempio **recv**, **read**). Così facendo è possibile "svuotare" la coda più velocemente.

Per molti amministratori la scelta più comune è quella di aumentare la profondità della coda. Questa sembra essere la soluzione più semplice ma al tempo stesso potrebbe non essere quella più idonea col tempo. Poichè nuove tecnologie di networking risultano essere sempre più veloci, le code continueranno a riempirsi sempre più velocemente. Ne consegue che sarà necessario modificare continuamente la profondità della coda in modo appropriato.

La soluzione ideale è quella di migliorare o configurare l'applicazione in modo da rimuovere più velocemente i dati del kernel, anche se tale operazione significa mettere in coda i dati nello spazio dell'applicazione. Questa operazione permette una archiviazione più flessibile dei dati, poichè essi potranno essere rimossi ed inseriti per soddisfare le esigenze dell'utente.

8.5. CONSIDERAZIONI MULTICAST

Quando applicazioni multiple sono in ascolto per un gruppo multicast, sarà necessario usare il codice del kernel che gestisce i frame multicast per duplicare i dati di rete per ogni socket. Questa operazione richiede tempo e si verifica in un contesto **softirq**.

L'aggiunta dei listener su un gruppo multicast ha un impatto diretto sul tempo di esecuzione del contesto **softirq**. L'aggiunta di un listener ad un gruppo implica la creazione da parte del kernel di una copia aggiuntiva per ogni frame ricevuto per quel gruppo.

Questo impatto è minimo in presenza di un traffico relativamente basso e con un numero di listener ristretto. Tuttavia con socket multipli in ascolto di un gruppo multicast con traffico elevato, un tempo più esteso di esecuzione del contesto **softirq** può risultare in una perdita di frame sia nella coda del socket che sulla scheda di rete. Tempi di esecuzione **softirq** maggiori riducono l'opportunità di esecuzione delle applicazioni in sistemi molto carichi, e la velocità con la quale saranno persi i frame multicast aumenterà a causa di un incremento del numero di applicazioni in ascolto ad un gruppo multicast con traffico elevato.

Per risolvere questo problema ottimizzare le code del socket ed i buffer hardware NIC come descritto in Sezione 8.4.2, «Coda del socket» o Sezione 8.4.1, «NIC Hardware Buffer» . Alternativamente ottimizzare l'uso del socket di una applicazione; per fare questo configurare l'applicazione in modo da controllare un singolo socket e ridistribuire i dati di rete ricevuti ad altri processi spazio-utente.

---

[4] Prendere in considerazione il cache affinity tra una CPU ed un NIC significa configurarli in modo da condividere la stessa cache L2. Per maggiori informazioni consultare Sezione 8.3, «Panoramica sulla ricezione dei pacchetti».

[5] Sezione 8.3, «Panoramica sulla ricezione dei pacchetti» contiene una panoramica sul percorso di un pacchetto, ciò potrà assistere l'utente nel determinare la posizione delle aree "problematiche" presenti in uno stack di rete.

[6] Il peso del dispositivo viene controllato tramite **`/proc/sys/net/core/dev_weight`**. Per maggiori informazioni sul peso e sulle implicazioni dovute alla sua modifica consultate Sezione 8.4.1, «NIC Hardware Buffer».

APPENDICE A. DIARIO DELLE REVISIONI

Revisione 4.0-22.1.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

Revisione 4.0-22.1 **Thu Apr 18 2013** **Chester Cheng**
I file della traduzione sono sincronizzati con con le versioni 4.0-22 dei sorgenti XML

Revisione 4.0-22 **Fri Feb 15 2013** **Laura Bailey**
Pubblicazione per Red Hat Enterprise Linux 6.4

Revisione 4.0-19 **Wed Jan 16 2013** **Laura Bailey**
Piccole correzioni (BZ#868404).

Revisione 4.0-18 **Tue Nov 27 2012** **Laura Bailey**
Pubblicazione per Red Hat Enterprise Linux 6.4 Beta.

Revisione 4.0-17 **Mon Nov 19 2012** **Laura Bailey**
Aggiunti suggerimenti SME re. sezione numad (BZ#868404).

Revisione 4.0-16 **Thu Nov 08 2012** **Laura Bailey**
Aggiunta una sezione bozza relativa a numad (BZ#868404).

Revisione 4.0-15 **Wed Oct 17 2012** **Laura Bailey**
Applicato suggerimento SME relativo alla discussione sull'eliminazione del blocco e spostata la sezione nelle Opzioni di montaggio (BZ#852990).
Aggiornate le descrizioni sul profilo delle prestazioni (BZ#858220).

Revisione 4.0-13 **Wed Oct 17 2012** **Laura Bailey**
Aggiornate le descrizioni sul profilo delle prestazioni (BZ#858220).

Revisione 4.0-12 **Tue Oct 16 2012** **Laura Bailey**
Migliorato l'uso del manuale (BZ#854082).
Corretta la definizione di ***`file-max`*** (BZ#854094).
Corretta la definizione di ***`threads-max`*** (BZ#856861).

Revisione 4.0-9 **Tue Oct 9 2012** **Laura Bailey**
Aggiunti i consigli FSTRIM al capitolo relativo ai File System (BZ#852990).
Aggiornata la descrizione di ***`threads-max`*** in base ai suggerimenti degli utenti (BZ#856861).
Aggiornata una nota sui miglioramenti per la gestione della frammentazione del GFS2 BZ#857782).

Revisione 4.0-6 **Thu Oct 4 2012** **Laura Bailey**
Aggiunta una nuova sezione all'utilità numastat (BZ#853274).

Revisione 4.0-3 **Tue Sep 18 2012** **Laura Bailey**
Aggiunta una nota re. nuove capacità perf (BZ#854082).
Corretta la descrizione del parametro file-max (BZ#854094).

Revisione 4.0-2 **Mon Sep 10 2012** **Laura Bailey**
Aggiunta una sezione BTRFS ed una introduzione di base al file system (BZ#852978).
Annotata l'integrazione di Valgrind con GDB (BZ#853279).

Revisione 3.0-15 **Thursday March 22 2012** **Laura Bailey**
Aggiunte ed aggiornate le descrizioni dei profili tuned-adm (BZ#803552).

Revisione 3.0-10 **Friday March 02 2012** **Laura Bailey**

Aggiornate le descrizioni dei parametri threads-max e file-max (BZ#752825).
Aggiornato il valore predefinito del parametro slice_idle (BZ#785054).

Revisione 3.0-8 **Thursday February 02 2012** **Laura Bailey**

Aggiunte e ristrutturate le informazioni sull'assegnazione della memoria e per il taskset e l'associazione della CPU con numactl su Sezione 4.1.2, «Ottimizzazione delle prestazioni della CPU» (BZ#639784).
Corretto l'uso dei link interni (BZ#786099).

Revisione 3.0-5 **Tuesday January 17 2012** **Laura Bailey**

Piccole correzioni a Sezione 5.3, «Come usare Valgrind per una analisi sull'uso della memoria» (BZ#639793).

Revisione 3.0-3 **Wednesday January 11 2012** **Laura Bailey**

Assicurata uniformità tra gli hyperlink interni ed esterni (BZ#752796).
Aggiunto Sezione 5.3, «Come usare Valgrind per una analisi sull'uso della memoria» (BZ#639793).
Aggiunto Sezione 4.1.2, «Ottimizzazione delle prestazioni della CPU» e riorganizzato Capitolo 4, *CPU* (BZ#639784).

Revisione 1.0-0 **Friday December 02 2011** **Laura Bailey**

Release per il GA di Red Hat Enterprise Linux 6.2.