# Università degli Studi di Pisa

## Facoltà di Scienze Matematiche Fisiche e Naturali

## Corso di Laurea specialistica in

## Tecnologie informatiche


Tesi di Laurea

# Individuazione di un hot spot in distributed hash table

**Relatori:**

*Prof. Massimo Coppola*

*Prof. Laura Ricci*

**Candidato:**

*Massimiliano La Gala*

**Anno Accademico: 2007/08**

*Alla mia famiglia*

# Indice generale

# Capitolo 1 - Introduzione

I servizi più diffusi sulle reti informatiche sono stati progettati con una struttura che distingue il fornitore del servizio dall'utilizzatore: protocolli come HTTP, POP3, IMAP, NNTP [1] sono soltanto alcuni esempi molto diffusi. Nella seconda metà degli anni novanta Napster[2], un programma per la condivisione di musica su internet, ha diffuso un diverso approccio per la costruzione di servizi sulle reti informatiche: gli utenti erano sia utilizzatori del servizio (scaricavano file musicali) che fornitori (permettevano agli altri utenti di scaricare i file musicali che avevano condiviso). Un'istanza di un programma che, come Napster, fornisce sia le funzionalità di server che di client è detta Peer ed il paradigma che lo sfrutta è chiamato Peer To Peer (P2P) [3] ovvero da pari a pari, per sottolineare l'eguale importanza che assume ogni nodo della rete.

La diffusione di questo nuovo paradigma ha permesso sia la realizzazione di nuove applicazioni, sia l'abbattimento dei costi di altre già esistenti. Per esempio, attualmente è possibile realizzare lo streaming video o la distribuzione di un file [4] mediante risorse molto limitate da parte del fornitore del servizio in confronto ad un'implementazione che utilizzi l'approccio client-server.

Il paradigma distribuito P2P offre notevoli vantaggi rispetto al tradizionale approccio client-server derivanti dall'assenza di un punto di centralizzazione che rende il sistema più tollerante ai guasti permettendone l'utilizzo anche se una parte consistente dei nodi della rete non è funzionante. Tuttavia operazioni che in un sistema centralizzato si eseguono semplicemente, come la memorizzazione, la ricerca, ed il recupero delle informazioni, in un sistema P2P risultano più complesse: ogni nodo della rete deve essere in grado di eseguire queste operazioni per permettere un corretto funzionamento delle applicazioni che la utilizzano a prescindere dalla disposizione fisica dei dati.

Le prime reti P2P fornivano una bassa scalabilità: Napster per esempio utilizza un server centralizzato per mantenere l'indice dei dati registrati nella rete. Il problema della scalabilità è stato affrontato con l'introduzione di una nuova categoria di reti P2P: le distributed hash table (DHT) [5] [6] [7] [8]. Ogni peer che partecipa ad una DHT è in grado di immettere dei dati in forma di una coppia (chiave,valore) e di recuperare una qualsiasi informazione immessa da un altro nodo. Le chiavi sono distribuite uniformemente in uno spazio logico, detto spazio degli identificatori, le cui caratteristiche sono determinate dalla DHT utilizzata.

La caratteristica più interessante di queste reti è la loro scalabilità: alcune reti DHT sono utilizzate ogni giorno da centinaia di migliaia di utenti [9]. Il consistent hashing [10], permette di distribuire il carico di lavoro in modo abbastanza equo fra tutti i peer: ogni nodo gestisce al massimo una quantità di chiavi che si discosta di un fattore logaritmico rispetto alla media[11].

L'importanza di una equa distribuzione del carico di lavoro fra i nodi è evidente: una distribuzione non equa del carico di lavoro può rallentare alcuni nodi, e nei casi peggiori può anche sovraccaricare il peer al punto di renderlo inutilizzabile, riducendo le prestazioni globali dell'intera rete DHT.

Per questo motivo sono stati sviluppati molti sistemi per migliorare la distribuzione del carico di lavoro fra i peer [12] [13] [14] [15]. Nel capitolo 2 sono presentati i principali sistemi di bilanciamento del carico utilizzati nelle DHT.

Le DHT sono state utilizzate in un numero crescente di campi applicativi, per esempio per realizzare un directory service distribuito per permettere ai partecipanti di una griglia di pubblicare le risorse che intendono mettere a disposizione degli altri nodi. In casi di utilizzo simile, la disponibilità di sole ricerche semplici, ovvero il recupero del valore associato ad una chiave è limitativo.

Per questo motivo, l'estensione delle DHT con sistemi più potenti di ricerca dei valori registrati nella rete è attualmente tema di ricerca: la possibilità di eseguire ricerche complesse come ad esempio la ricerca di tutte le chiavi compresi fra due valori, oppure delle ricerche che selezionano i dati restituiti in base a più attributi è ancora oggetto di studio.

Il consistent hashing, il sistema che bilancia il carico delle DHT mediante la distribuzione delle chiavi nello spazio degli identificatori, ha l'effetto collaterale di distruggere la località dei dati: due chiavi associate a valori molto simili possono essere assegnate a posizioni della DHT molto lontane fra di loro, complicando le operazioni necessarie per poter eseguire query complesse.

Questo problema è stato affrontato mediante la modifica del sistema di distribuzione dei dati in modo da preservare, anche solo parzialmente, la località dei dati immessi nella rete [16] [17].

L'effetto di questa modifica è che la distribuzione generata delle chiavi è dipendente in qualche misura dalla distribuzione dei valori ad esse associate, riducendo la capacità del consistent hashing di distribuire uniformemente le chiavi nello spazio degli identificatori e quindi di bilanciare il carico di lavoro fra i peer.

In questi sistemi, un sistema di bilanciamento del carico che agisca sulla distribuzione delle chiavi è difficilmente realizzabile, in quanto da essa dipende la conservazione della località delle chiavi, per cui è necessario agire sulla distribuzione dei peer.

In questo lavoro di tesi, propongo un sistema di individuazione dei nodi sovraccarichi di una DHT basato sull'analisi dinamica del traffico inoltrato dai peer. In questo sistema, ogni peer della rete analizza il traffico che lo attraversa per eseguire una stima del carico delle sezioni dello spazio degli identificatori a cui è connesso: la stima si basa sul confronto fra il numero di query instradate verso una sezione dello spazio degli identificatori e la sua speranza matematica.

La costruzione di un modello che permetta di calcolare la speranza matematica di ogni sezione dello spazio degli identificatori è stato un punto focale del lavoro di tesi, ed è stato eseguito tramite l'analisi di simulazioni appositamente realizzate con lo scopo di individuare le caratteristiche del modello.

La capacità di un nodo di stimare il livello di carico di una sezione dello spazio degli identificatori ha permesso la definizione di un algoritmo di ricerca dei nodi sovraccarichi. Questa ricerca è eseguita tramite un routing guidato dalle stime di carico eseguite dai peer: un nodo instrada il messaggio verso un peer che

ritiene posizionato in una sezione dello spazio degli identificatori particolarmente carica.

Un tale sistema di individuazione dei nodi sovraccarichi può essere utilizzato in un sistema di bilanciamento del carico che inserisca i nodi che richiedono di far parte della rete DHT in modo da ridurre il carico di lavoro del nodo sovraccarico individuato.

La struttura della tesi è la seguente. Nel capitolo 2 è presentata una panoramica di alcune reti DHT e di alcuni sistemi di bilanciamento del carico presenti in letteratura. Nel capitolo 3 è proposta una nuova definizione del carico di lavoro sui nodi di una DHT, si presenta il sistema di individuazione delle sezioni dello spazio degli identificatori sovraccarico, e la metodologia utilizzata per determinare lo stato di carico di un nodo. Il capitolo 4 presenta le simulazioni eseguite per estrapolare la funzione per il calcolo della speranza matematica del numero di query che raggiungono una sezione dello spazio degli identificatori. Nel capitolo 5 sono presentati gli strumenti utilizzati per eseguire le simulazioni. Nel capitolo 6 si propone l'algoritmo di individuazione dei nodi sovraccarichi ed i test eseguiti su di esso.

# Capitolo 2 - Bilanciamento del carico su DHT: stato dell'arte

I primi sistemi P2P soffrono di una limitata capacità nel reperire le informazioni registrate, Napster[2] per esempio utilizza un server centrale per mantenere gli indici dei file disponibili nei client, soluzione che limita la scalabilità del sistema. Gnutella [18], un'altra applicazione P2P dello stesso periodo, segue un approccio diverso: si basa su un sistema completamente distribuito ottenuto tramite la rimozione dell'indice dei file registrati. L'operazione di ricerca è effettuata tramite flooding di una porzione della rete, questa operazione non è però in grado di garantire il reperimento dell'informazione cercata anche se essa è presente nella rete. Questa mancanza è causata dalla necessità di ridurre l'utilizzo di risorse durante l'operazione di ricerca, per cui si è limitata la distanza massima entro la quale un peer può essere contattato: l'operazione di ricerca copre soltanto una porzione dei nodi della rete, per cui un dato registrato in un nodo non contattato non può essere recuperato.

Nell'anno 2001 è stato proposto un approccio diverso che permette di superare i limiti di reperibilità delle informazioni contenuti nella rete P2P tipici dei primi sistemi: questi sistemi [8] [6] [7] [5] detti *distributed hash table* (DHT) forniscono un'operazione di ricerca dei dati che garantisce di trovare l'informazione cercata se presente nella rete, l'esecuzione di questa operazione è veloce e scalabile in quanto è un' operazione distribuita che utilizza poche risorse.

## 2.1 - Distributed hash table

Le funzionalità delle DHT sono del tutto simili a quelle fornite da una hash table: gestiscono le informazioni come coppie di chiavi e valori, i primi utilizzati per registrare e ricercare i secondi, ovvero i dati da inserire nella rete. Le DHT sono anche in grado di distribuire più o meno omogeneamente la gestione delle chiavi fra i nodi che partecipano alla rete secondo un preciso algoritmo noto a tutti i peer, per cui ad ogni nodo è affidata la gestione di chiavi provenienti potenzialmente da ogni nodo della rete.

La gestione di ogni chiave comporta l'utilizzo di alcune risorse, come ad esempio memoria per immagazzinarla, CPU e banda per rispondere alle richieste da parte degli altri nodi. In base all'utilizzo che si fa della DHT, una risorsa può essere più importante rispetto alle altre. Per esempio in una rete utilizzata come Grid information services, dato l'elevato numero di aggiornamenti e di query, risorse come banda utilizzata e tempo di CPU utilizzata possono essere molto rilevanti; mentre in una DHT il cui utilizzo è di registrare dati di grosse dimensioni, acquisisce maggiore rilevanza la quantità di chiavi registrate e la quantità di spazio di memorizzazione richiesto. La quantità totale delle risorse utilizzate da un nodo per la gestione delle chiavi a lui assegnate determina il suo carico di lavoro.

Se un nodo dovesse gestire una quantità di dati notevolmente superiore rispetto a quella degli altri partecipanti della rete, si avrebbe uno sbilanciamento del carico di lavoro, che causerebbe un cattivo utilizzo delle risorse a disposizione, portando a un rallentamento dei tempi di servizio e quindi un degradamento delle performance della rete DHT.

Nei casi di sbilanciamento peggiori alcuni nodi potrebbero addirittura non riuscire a reggere il carico di lavoro a cui sono sottoposti, rendendosi incapaci di rispondere alle richieste che provengono dagli altri nodi.

La capacità di distribuire equamente i dati fra i partecipanti è quindi un punto cardine di un sistema DHT che ne determina caratteristiche fondamentali per un sistema distribuito, come la scalabilità e l'efficienza dell'intero sistema.

Le reti DHT sono state sviluppate mettendo in primo piano il concetto di bilanciamento del carico di lavoro sui singoli nodi. Per questo scopo, hanno utilizzato un sistema che assicura una buona distribuzione delle chiavi sui nodi: il consistent hashing.

### 2.1.1 - Consistent Hashing

I sistemi DHT proposti nel 2001 [8] [6] [7] [5] si basano sul consistent hashing, un precedente lavoro [10] del 1997 che definisce un protocollo per distribuire le richieste di servizio di un server web su un numero variabile di macchine di cache, allo scopo di bilanciare il carico di lavoro su ogni macchina utilizzata.

Nel modello utilizzato nell'articolo [10] si suppone che ogni macchina di cache possa mandare un messaggio ad ogni altra macchina. Il modello specifica che il numero di macchine di cache a disposizione del sistema è un numero variabile che non è noto né ai client che utilizzano il sistema, né alle altre macchine di cache.

L'obiettivo è quello di evitare con “alta probabilità” che un web server o una macchina di cache non siano in grado di rispondere alle richieste per l'eccessivo carico di lavoro. La definizione di “alta probabilità” è : probabilità superiore a $1-\frac{1}{N}$ dove N è un parametro scelto in base al sistema.

Il consistent hashing bilancia il carico del server distribuendo i dati nelle macchine cache, operazione eseguita tramite l'utilizzo di una funzione di hash: il risultato del hashing sull'identificatore del dato identifica la macchina di cache a cui il dato deve essere assegnato.

I client per recuperare questi dati dovranno a loro volta applicare la funzione di hash sull'identificatore per selezionare la macchina di cache a cui è stato assegnato il dato e procedere così alla richiesta.

Una classica funzione di hash come $x \to ax+b(mod\ p)$ non è però utilizzabile a causa della variabilità del numero di macchine di cache: l'introduzione o la rimozione di una macchina di cache causerebbe la necessità di dover rimappare la quasi totalità dei dati rendendo inutili tutti i dati registrati nelle cache.

Il consistent hashing per affrontare questo problema deve gestire le diverse "viste" dei client, ovvero il sottoinsieme di macchine di cache di cui un determinato client è a conoscenza. Ciò è effettuato tramite l'introduzione delle *ranged hash function,* funzioni di hash che per ogni vista associano un oggetto ad una macchina di cache.

Un client che vuole richiedere un dato utilizza la funzione di hash per selezionare la macchina di cache appartenente alla vista del nodo a cui richiedere il dato ricercato. La macchina di cache provvederà ad acquisire tale dato e servire la richiesta.

Nell'articolo [19] sono definite le proprietà che permettono di individuare una buona *ranged hash function*:

- **Bilanciamento**: Una famiglia di ranged hash function è bilanciata se, data una particolare vista V, un insieme di elementi ed una ranged hash function scelta casualmente nella famiglia di funzioni, si ha un'alta probabilità che il numero di elementi mappati in ogni macchina di cache sia $O\left(\frac{1}{|(V)|}\right)$

- **Monotonicità**: una ranged hash function è monotona se per ogni coppia di viste $V_1$ e $V_2$ , legate dalla relazione $V_1 \subseteq V_2$ è verificato che $F_{v_2}(i) \in V_1 \to F_{v_2}(i) = F_{v_1}(i)$ dove *i* è un qualsiasi elemento da mappare. Questa proprietà asserisce che se degli elementi sono originalmente assegnati ad una vista $V_1$ nella quale vengono introdotti nuove possibili macchine di cache formando $V_2$ , allora agli elementi è permesso di muoversi da una vecchia macchina di cache verso una nuova, ma non da tra due vecchie macchine di cache .

- **Spread**: sia $V_1 \dots V_v$ un insieme di viste, contenenti complessivamente $C$ macchine di cache e singolarmente almeno $\frac{C}{t}$ posizioni, dove $t$ è il numero minimo di macchine di cache note a tutti. Lo spread $\sigma(i)$ dell'elemento *i* è la quantità $\left| \left\{ F_{v_j}(i) \right\}_{j=1}^{V} \right|$ . Lo spread di una funzione di hash $\sigma(f)$ è il massimo spread degli elementi. Questa caratteristica descrive il numero di macchine di cache diverse a cui un elemento può essere assegnato a causa delle diverse viste. Una buona funzione di consistent hashing deve avere uno spread basso.

- **Carico**: sia definito un insieme di viste V come fatto precedentemente. Per una determinata ranged hash function f e una posizione p il carico $\lambda(p)$ è uguale al totale degli oggetti assegnati alla macchina di cache p per tutte le viste di V. Una buona consistent hash function dovrebbe avere un basso valore di carico.

Nell'articolo del consistent hashing [10], è presentata una famiglia di ranged hash function che possiede buoni valori di bilanciamento, monotonicità, spread e carico. Su questa famiglia di funzioni di hash si basa il sistema di bilanciamento del carico delle reti DHT.

## 2.2 - Consistent hashing nelle DHT

Le reti DHT [8] [6] [7] [5], per distribuire i dati sui nodi della loro rete, utilizzano una applicazione del consistent hashing descritto nel paragrafo precedente.

Il consistent hashing utilizzato nelle DHT si basa sull'utilizzo di uno spazio logico degli identificatori sul quale è definita una funzione $\delta(id_{1,} id_2)$ che permette di determinare la distanza fra due identificatori. A questa funzione se ne aggiungono altre due : la prima $f : Nodo \rightarrow Identificatore$ che mappa i nodi nello spazio degli identificatori, e la seconda $g : chiave \rightarrow Identificatore$ che data la chiave genera l'identificatore ad essa associato.

Alle funzioni $f$ e $g$ è richiesto che la distribuzione degli identificatori da loro fornita sia associabile ad una distribuzione uniforme generata pseudo-casualmente.

L'assegnamento delle chiavi ai nodi avviene tramite l'utilizzo della funzione di distanza $\delta$ : le chiavi sono assegnate al peer a loro più vicino.

Un nodo che richiede di partecipare alla rete DHT tramite l'operazione di JOIN, deve ottenere un identificatore che determini il punto di inserimento del nodo, perciò la funzione $f : Nodo \rightarrow Identificatore$ è una parte importante di questa operazione.

L'identificatore fornito da $f$ determina indirettamente quali e quante chiavi saranno gestite da quel nodo: l'impatto sul carico di lavoro che dovrà essere gestito dal peer è notevole. Per questo motivo sono state avanzate molte proposte [12] [13] [14] di algoritmi di JOIN da utilizzare per migliorare il bilanciamento del carico fra i nodi della rete. Questi algoritmi seguono uno schema simile e si differenziano soltanto per alcune caratteristiche, come ad esempio la dimensione dell'insieme dei punti fra i quali scegliere l'ID del nodo.

L'operazione di inserimento di un nodo può essere descritta mediante una sequenza di passi generici comuni a quasi tutte le DHT:

1. il nuovo nodo $P$ che vuole inserirsi nella rete genera $j \geq 1$ punti nello spazio degli identificatori secondo un algoritmo prefissato.

2. Contatta un nodo $Q$ noto a priori e, tramite esso, esegue l'operazione di ricerca degli identificatori selezionati precedentemente al fine di determinarne i nodi che li gestiscono.

3. Il nodo $P$ scegli $k$ punti dello spazio degli identificatori con $1 \leq k \leq j$ in cui inserirsi, prelevando parte della gestione dello spazio delle chiavi ai nodi contattati secondo un certo algoritmo. I $k$ punti scelti saranno gli ID del nodo.

Dai passi descritti per l'operazione di inserimento, si può notare che l'operazione più costosa in termini di messaggi spediti è la ricerca dei j punti, dunque il costo dell'inserimento di un nodo è pari a quello della ricerca delle j chiavi. Considerando le caratteristiche delle DHT, il costo di una singola ricerca è generalmente $O(\log(n))$ , per cui il costo complessivo dell'inserimento di un nodo è $O(j * \log(n))$ .

Nella maggior parte dei sistemi DHT, sia per la funzione $f : Nodo \rightarrow Identificatore$ che assegna l'identificatore ai nodi, sia per la funzione $g : chiave \rightarrow Identificatore$ che assegna l'identificatore ad una chiave, si utilizza una funzione di hash, come ad esempio la SHA1, applicata alla risorsa di cui si vuole calcolare l'identificatore.

I peer dunque si assegnano autonomamente il proprio identificatore calcolando, per esempio, l'hash del proprio indirizzo IP + porta TCP sulla quale il programma è in ascolto.

La caratteristica del consistent hashing di ridurre al minimo le conseguenze dovute all'inserimento o la rimozione di un peer è mantenuta: la modica risulta infatti essere locale interessando solamente i nodi la cui posizione logica li rende limitrofi al peer che è entrato a far parte o ha abbandonato la rete, lasciando inalterati tutti gli altri nodi.

### 2.2.1 - Chord

Chord [7] è una DHT che definisce un protocollo che supporta un'unica operazione: mappare una data chiave in un nodo appartenente alla rete. Alle applicazioni che implementano questo protocollo è lasciata la scelta di come gestire questa informazione, ad esempio i nodi potrebbero salvare un valore associato alla chiave per gestire eventuali future richieste.

Chord, per eseguire l'associazione della chiave al nodo, utilizza una variante del Consistent hashing. In questo modo può avvantaggiarsi delle capacità di bilanciamento del carico da esso fornite: ad ogni peer sarà assegnato un numero di chiavi ben distribuito evitando i nodi eccessivamente sovraccarichi. Inoltre rende efficienti le operazioni di join e di leave eseguite dai peer della rete, a causa della località di queste operazioni.

L'assegnamento dell'identificatore logico (ID) ai nodi ed alle chiavi è un'operazione distribuita compiuta autonomamente dal peer che necessita di questa informazione: è effettuata, in entrambi i casi, tramite l'utilizzo di una normale funzione di hash come ad esempio SHA-1. L'ID di un nodo è calcolato autonomamente tramite l'utilizzo di hashing dell'indirizzo IP, mentre l'identificatore di una chiave è calcolato tramite hashing della stessa chiave.

Lo spazio degli identificatori è organizzato come un anello modulo $2^m$ , per cui la funzione di distanza fra due identificatori utilizzata è $\delta : (id_1 - id_2 + 2^m) \bmod 2^m$ : le chiavi sono assegnate al primo nodo il cui ID è uguale o maggiore del ID della chiave.

I nodi che partecipano alla DHT gestiscono una struttura dati distribuita, la routing table, che permette di risolvere efficientemente l'operazione di ricerca di una chiave. A questo scopo ogni peer mantiene una quantità di informazioni su un numero limitato ma ben definito di nodi che partecipano alla rete.

L'assenza di punti di centralizzazione nella routing table permette a Chord di fornire un' ottima scalabilità in termini di numero di peer che partecipano alla rete.

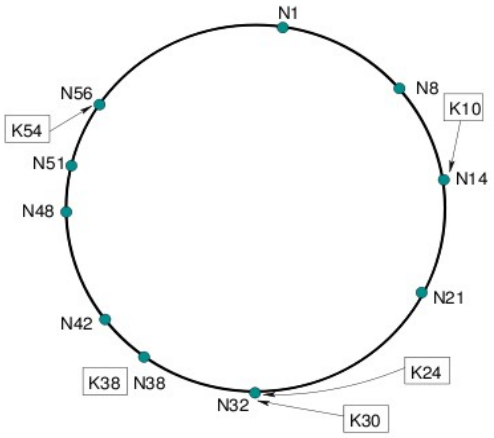


*Illustrazione 1: rappresentazione dello spazio degli identificatori di una rete Chord con 10 nodi e 5 chiavi: è mostrato l'assegnamento delle chiavi ai nodi*

In Chord la porzione della routing table locale ad un nodo è chiamata finger table. Questa struttura contiene al massimo informazioni relative a $m$ nodi diversi, dove $2^m$ è il numero massimo di chiavi distinte che possono essere inserite nella rete. $m$ è un numero il cui valore è generalmente $128$ o $160$.

La finger table di ogni nodo è un array, che in ogni casella contiene un collegamento ad un altro nodo noto con la seguente caratteristica: nella $i$ - esima casella vi è inserito il collegamento al primo nodo noto il cui ID supera l'identificatore del nodo corrente di almeno $2^{i-1}$. Nelle caselle della finger table sono mantenuti, oltre all'ID del nodo, anche tutti i dati necessari per stabilire un collegamento diretto con quel nodo, come per esempio il suo indirizzo IP e la porta TCP sulla quale il programma è in ascolto.

La ricerca di una chiave segue un algoritmo distribuito fra alcuni nodi che sono man mano sempre più vicini alla destinazione. L'algoritmo prevede che un nodo con il compito di gestire una ricerca di una chiave, debba prima controllare se è il nodo a cui ne è stata affidata la gestione, in tal caso la ricerca termina e si comunica il risultato della query al nodo che l'ha generata. Se invece il nodo non gestisce la chiave, si inoltra la query al nodo noto più vicino alla destinazione.

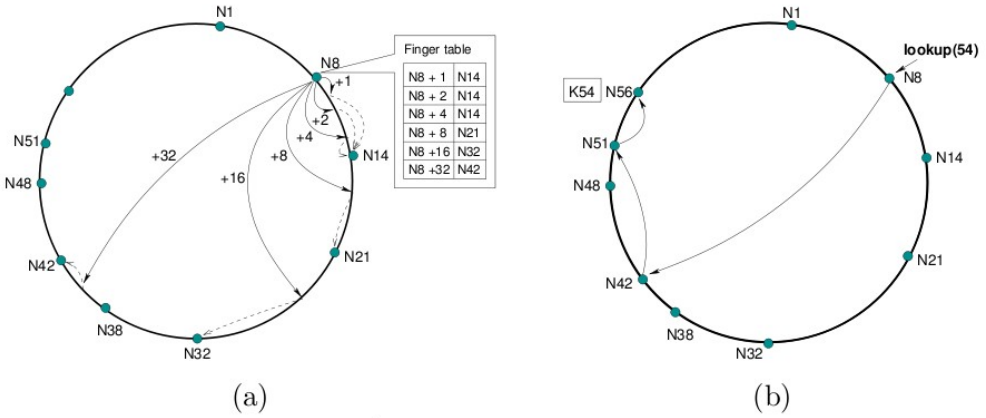


*Illustrazione 2: a) finger table del nodo 8. B) Il percorso effettuato dalla query della chiave 54 che comincia nel nodo 8*

L'individuazione del nodo noto più vicino è eseguita utilizzando la finger table: il peer più vicino alla destinazione è presente nella casella di questa struttura con indice maggiore il cui ID non supera quello della chiave.

L'efficienza del routing in Chord dipende dall'integrità dei dati della routing table: durante l'inserimento o l'uscita di un nodo dalla rete, la routing table necessita di un lasso di tempo per includere le variazioni apportate dalla modifica dell'insieme dei nodi che fanno parte della rete. Una rete Chord, la cui routing table non contiene incoerenze è detta stabile.

In una rete Chord stabile, ogni volta che il routing esegue un hop, la distanza fra il nodo che gestisce correntemente la query e la sua destinazione si dimezza. Questa caratteristica ha un impatto molto importante sulle prestazioni della rete, in quanto implica che il numero di peer contattati per eseguire una ricerca cresca in modo logaritmico rispetto al numero di nodi presenti sulla rete.

### 2.2.2 - Pastry

Un sistema Pastry [6] è una DHT che definisce un overlay network di nodi che per la sua organizzazione utilizza una diversa geometria dello spazio degli identificatori rispetto a Chord.

Ogni nodo della rete Pastry è contraddistinto da un nodeID, ovvero un identificatore di 128 bit assegnato all'interno di uno spazio con struttura circolare i cui valori possibili variano fra0 e $2^{128}-1$ . L'operazione di assegnamento del nodeID è completamente distribuita: viene effettuata dallo stesso nodo prima di richiedere l'ingresso alla rete. La funzione utilizzata per questo scopo deve distribuire i nodeID assegnati in modo uniforme sull'intero spazio degli identificatori, per questo scopo si utilizza una funzione di hash come SHA-1 applicata o ad una chiave pubblica che contraddistingue il nodo, oppure al suo indirizzo di rete. La medesima funzione di hash è utilizzata per assegnare un identificatore alle chiavi in modo da distribuirle uniformemente sull'intero spazio

Tabella di routing per il nodo n=65A1FC04

| 0x | 1x | 2x | 3x | 4x | 5x | | 7x | 8x | 9x | Ax | Bx | Cx | Dx | Ex | Fx |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 60x | 61x | 62x | 63x | 64x | | 66x | 67x | 68x | 69x | 6Ax | 6Bx | 6Cx | 6Dx | 6Ex | 6Fx |
| 650x | 651x | 652x | 653x | 654x | 655x | 656x | 657x | 658x | 659x | | 65Bx | 65Cx | 65Dx | 65Ex | 65Fx |
| | | | | | | | | | | | | | | | |

*Illustrazione 3: esempio di tabella di routing per il nodo Pastry con nodeID 65A1FC04 con b = 4. Nella riga 0 sono inseriti i riferimenti ai nodi che non condividono un prefisso con il nodo n, nella riga 1 sono presenti i nodi il cui nodeID comincia per 6, nella riga 2 sono presenti i nodi il cui nodeID comincia per 65*

degli identificatori.

La gestione di una chiave è affidata al nodo secondo la seguente funzione di distanza $\delta : \left|((id_1 - id_2 + 2^m) \bmod 2^m)\right|$ , ovvero il nodo il cui ID minimizza la distanza numerica dal ID della chiave.

Gli identificatori, sia dei nodi che delle chiavi, pur essendo delle sequenze di 128 bit vengono trattate come sequenze di cifre di $2^b$ bit. Questa suddivisione logica degli identificatori è utilizzata da pastry per scegliere il sottoinsieme di nodi di cui mantenere i dati per popolare la routing table. Questa struttura è costituita da una matrice con $\lceil(\log_{2^b})\rceil$ righe ognuna contenente $2^b-1$ posizioni nella quale sono mantenuti i riferimenti ai nodi noti della rete secondo la seguente organizzazione: nella casella di posizione $(n,i)$ si inserisce il collegamento ad un nodo il cui nodeID condivide con il nodo corrente esattamente le prime n cifre, ed il valore della cifra $n+1$ è $i$ . Con questa struttura della routing table un peer manterrà un numero basso di collegamenti verso i nodi il cui nodeID è distante dal proprio, che aumenteranno in modo esponenziale man mano che il nodeID dei peer diventa numericamente più vicino al proprio.

Dall'analisi della routing table si capisce l'importanza del parametro $b$ : da esso dipende la forma della tabella di routing e dunque determina il numero medio di elementi presenti nella tabella e la lunghezza media del routing. È dunque necessario scegliere un compromesso fra questi due valori: la dimensione media riempita della routing table è approssimativamente $\lceil\log_{2^b}(N)*(2^b-1)\rceil$ , mentre il numero di hop necessari per completare la ricerca di un identificatore è $\lceil\log_{2^b}(N)\rceil$ . Un valore utilizzato comunemente per $b$ è 4, per cui in una rete con $10^6$ nodi si ottengono una media di 75 entry nella tabella di routing ed il numero medio di hop è 5, mentre in una rete con $10^9$ nodi la routing table conterrà una media di 105 entry ed un routing che richiede 7 hop.

L'algoritmo di routing di pastry, oltre all'utilizzo della routing table, necessita di un'ulteriore struttura dati detta leaf set nella quale sono mantenute le informazioni dei peer più vicini al nodo secondo la funzione di distanza $\delta:\lceil((id_1-id_2+2^m)\,mod\,2^m)\rceil$ . Essa contiene $L$ elementi, di cui $L/2$ sono riservati ai nodi il cui nodeID è maggiore di quello del nodo, ed i restanti $L/2$ sono riservati ai nodi il cui nodeID è minore. Questa struttura è anche utilizzata per aumentare la consistenza della rete e per prevenire perdite di dati in caso di

fallimento da parte di un nodo, infatti in quelli che fanno parte del leaf set di un nodo replicano le chiavi da esso gestite.

L'algoritmo di routing utilizzato da pastry è un algoritmo distribuito che coinvolge un piccolo insieme di nodi partecipanti alla rete. Un nodo che vuole ricercare un identificatore, controlla se questo è contenuto all'interno del suo leaf set, in tal caso inoltra la richiesta di routing al nodo che gestisce la chiave. In caso contrario viene esaminata la routing table: il nodo scelto per continuare l'operazione di ricerca è quello numericamente più vicino alla destinazione, ovvero il nodo il cui nodeID condivide con l'identificatore il prefisso più lungo e la prima cifra diversa è quella numericamente più vicina.

Ad ogni passo del routing, il prefisso in comune fra l'identificatore del nodo che gestisce la query e l'identificatore da ricercare si allunga, per cui il numero di hop da attraversare prima di trovare la destinazione della query è proporzionale al numero di cifre dell'identificatore, ovvero il costo di una query sarà $O(\log_{2^b}(N))$ .

Un peer di Pastry mantiene un' ulteriore struttura non utilizzata dal routing il cui scopo è quello di ottimizzare la overlay network. Un nodo che costruisce la propria routing table ha la possibilità di scegliere quale nodo inserire nella casella $(n, i)$ , infatti spesso esiste più di un nodo che ne ha i requisiti. L'ottimizzazione della overlay network si basa sulla scelta di una funzione che permette di definire la “distanza” al livello di rete IP fra due nodi, come ad esempi il round trip time o il numero di router da attraversare per raggiungere un nodo, diminuendo così i tempi complessivi di comunicazione fra i peer.

## 2.3 - consistent hashing: limitazioni

Il consistent hashing ha permesso lo sviluppo della tecnologia DHT fornendo un sistema poco costoso ed efficiente di bilanciamento del carico di lavoro fra i nodi, ma questa tecnica non è perfetta a causa di alcune assunzioni intrinseche che ne diminuiscono l'efficacia. La presenza di queste imperfezioni è stato motivo di studio[11] [12] [13] [14] [15] al fine di migliorare la capacità di bilanciamento del carico fornita.

La prima assunzione del consistent hashing i cui effetti influiscono sulla capacità di bilanciare il carico è di associare la medesima quantità di carico ad ogni chiave presente nella DHT: nel consistent hashing è assente un meccanismo che permetta di differenziare le chiavi in base alla quantità reale di risorse che un nodo utilizza per la loro gestione.

Per esempio, in una rete in cui a tutti i nodi è assegnata la medesima quantità di chiavi, condizione ottima di distribuzione del carico secondo i parametri del consistent hashing, non è assicurato che ogni nodo sia sottoposto al medesimo carico di lavoro: il numero di query che ogni nodo deve servire può variare considerevolmente in base alla popolarità delle chiavi che gestisce.

Il consistent hashing affronta questo problema solamente in termini statistici, in quanto la distribuzione casuale delle chiavi rende improbabile che queste siano assegnate al medesimo nodo. Nel caso in cui il numero di chiavi non sia sufficiente per fornire i termini statistici richiesti, allora il consistent hashing non è in grado di fornire una buona distribuzione del carico.

La seconda assunzione è quella di considerare uguali tutti i nodi, non differenziandoli in base alle risorse di calcolo di cui dispongono o che desiderano utilizzare per la gestione della rete. Per esempio anche in presenza di una perfetta distribuzione del carico di lavoro fra i peer, è possibile che alcuni risultino sovraccarichi a causa delle limitate risorse hardware di cui dispongono come un processore lento, poca RAM o collegamento alla rete con banda limitata.

Il consistent hashing non fornisce una distribuzione ottima delle chiavi sui nodi nemmeno nell'ipotesi che le due precedenti assunzioni siano rispettate.

L'Illustrazione 4 mostra la distribuzione di 500.000 chiavi su 4096 nodi di una rete Chord, con la linea grigia è indicato il punto di ottimo che corrisponde a circa 122 chiavi per nodo.

E' possibile notare che la distribuzione non è concentrata intorno all'ottimo, garantendo così una equa distribuzione del numero di chiavi assegnate ai nodi , ma al contrario sono presenti sia un elevato numero di nodi che gestiscono un basso numero di chiavi (picco vicino allo 0) sia un elevato numero di peer che gestiscono un elevato numero di chiavi (la lunga "coda" a destra della distribuzione).

In [11] è mostrato che il rapporto fra la dimensione dello spazio degli identificatori assegnato ad un nodo è con elevata probabilità $\Theta(\log(n))$ volte la dimensione dello spazio degli identificatori medio.

L'Illustrazione 5 mostra il numero di nodi a cui non è assegnata la gestione di nessuna chiave al variare del numero di chiavi nella rete. Si può notare come il numero di nodi senza documenti è un numero non nullo anche in presenza di un milione di chiavi, e che il numero di nodi senza chiavi è consistente per un numero di chiavi non elevato.

La presenza di un numero così elevato di nodi che non gestiscono alcuna chiave è stata attribuita alla tecnica di divisione dello spazio delle chiavi durante l'inserimento di un nuovo nodo nella DHT. La determinazione del punto in cui eseguire l'inserimento del nuovo nodo e dunque nel quale dividere lo spazio delle chiavi è eseguita utilizzando la funzione di hash, che a causa della sua caratteristiche di generare numeri pseudo-casuali garantisce ad ogni punto la medesima probabilità di essere scelto.

Se si suddivide lo spazio degli indirizzi assegnato ad un nodo basandosi esclusivamente sull'identificatore generato dalla funzione di hash, è probabile che quest'ultimo sia vicino ad uno degli estremi dello spazio delle chiavi: il risultato

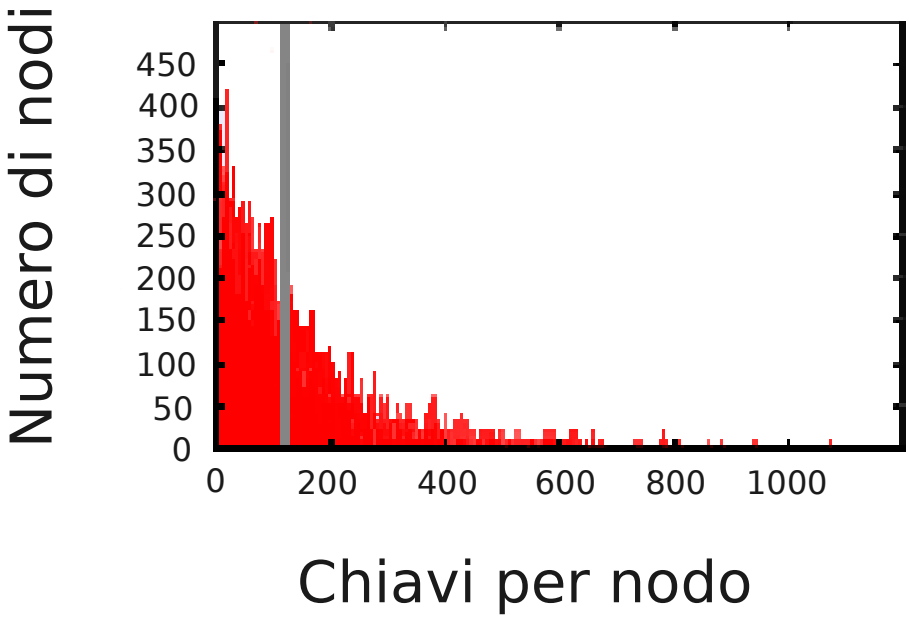


*Illustrazione 4: Distribuzione di 500.000 chiavi su 4096 nodi. La linea grigia indica l'ottimo: circa 122 documenti per nodo.*

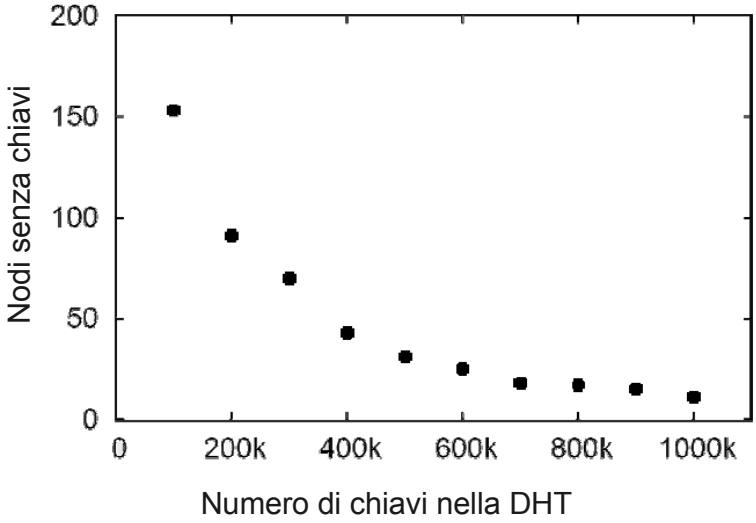


*Illustrazione 5: Numero di nodi senza carico in una rete chord con M = 22 e numero di documenti da 100.000 a 1.000.000*

della ripartizione saranno due spazi di dimensione diversa di cui uno può essere di dimensione molto ridotta.

In [12] si mostra come in una rete in cui gli $n$ nodi sono inseriti casualmente, la dimensione della più grande sezione dello spazio degli identificatori sia $O(\frac{\log(n)}{n})$ mentre la dimensione media è di $\frac{1}{n}$, ovvero un nodo può arrivare a gestire uno spazio degli identificatori più grande della media di un fattore logaritmico.

Dato che la probabilità di un nodo di ricevere l'assegnazione di una nuova chiave è direttamente proporzionale alla dimensione dello spazio degli identificatori ad esso assegnato, un nodo che gestisce uno spazio degli identificatori di dimensione estremamente ridotta ha un'elevata probabilità di non ottenere la gestione di nessuna chiave. Si è dunque ipotizzato che una distribuzione più equa fra i nodi dello spazio delle chiavi possa migliorare il bilanciamento del carico.

Su questo principio sono state sviluppate molte tecniche che cercano di ridurre lo sbilanciamento modificando le modalità in cui avviene l'inserimento di un nodo all'interno della rete DHT.

### 2.3.1 - power of two choices

Byers ed altri in [12] propongono di bilanciare il carico intervenendo sul processo di assegnamento di chiavi ai nodi, tramite un processo di selezione dei nodi che permette di evitare che una chiave venga assegnata ad un nodo sovraccarico in favore di uno più scarico. L'operazione di assegnamento della chiave avviene scegliendo il nodo più scarico di un piccolo insieme di nodi che sono legati tramite un algoritmo a quella chiave.

Questa operazione, per non deteriorare l'efficienza della rete DHT, non deve rallentare il processo di reperimento di una chiave , per cui è necessario che l'individuazione degli identificatori dei nodi a cui potrebbe essere assegnata la chiave avvenga in locale.

L'algoritmo proposto da Byers in [12] ed altri utilizza $d+1$ funzioni di hash con $d \geq 2$ concordate a priori fra tutti i peer. La prima di queste funzioni (la $f_0$ ) è utilizzata esclusivamente per inserire i nodi nella rete, mentre le restanti $d$ funzioni di hash sono utilizzate esclusivamente per l'inserimento ed il reperimento delle chiavi nella DHT.

Quando un nodo vuole pubblicare un nuovo documento, calcola il valore risultante di tutte le $d$ funzioni hash, ed esegue $d$ ricerche parallele per individuare quali nodi gestiscono i punti ottenuti. Dopo l'interrogazione del carico di questi nodi, la gestione della chiave verrà affidata a quello che vanta il carico di lavoro minore.

Per poter individuare il nodo a cui è assegnata la gestione di una chiave, permettendo così di recuperare il valore ad esso associato sono state avanzate due proposte.

La prima, più semplice, consiste nell'eseguire la ricerca del documento utilizzando tutte le funzioni di hash a disposizione: $d-1$ ricerche otterranno un risultato negativo,mentre una sola restituirà la chiave ricercata. Sebbene queste ricerche possano essere eseguite parallelamente, l'incremento del traffico di rete generato sarebbe considerevole: la ricerca di ogni singolo documento produrrebbe il traffico di $d$ singole ricerche.

Per ovviare a questo problema, Byers ed altri propongono una seconda tecnica di ricerca basata sull'adozione dei puntatori. L'operazione di inserimento di una chiave è modificata in modo che i $d-1$ nodi che non hanno ottenuto la gestione della chiave, registrino al suo posto un puntatore verso il nodo a cui è stata affidata la gestione della chiave.

Il costo dell'operazione di ricerca della chiave è notevolmente ridotto rispetto al caso precedente, infatti sarà sufficiente selezionare una qualsiasi delle funzioni di hash ed eseguire una singola ricerca: il risultato sarà o la chiave, o il puntatore al nodo che gestisce la chiave.

L'introduzione dei puntatori, pur riducendo notevolmente il traffico di rete, presenta comunque degli svantaggi dovuti al loro costo di gestione: il nodo a cui è

assegnata la chiave potrebbe fallire a causa di un guasto o più semplicemente uscire dalla rete rendendo incoerenti i puntatori presenti. Per cui risulta necessario aggiornare periodicamente i puntatori, permettendo al documento di rimanere rintracciabile.

L'utilizzo di power of two choices produce una sensibile riduzione del carico massimo che è assegnato ad un nodo: garantisce infatti che il numero massimo di chiavi assegnate ad un peer sia ridotto da $O(\frac{\log(n)}{n})$ a $\frac{\log(\log(n))}{\log(d)}+O(1)$ dove d è il numero di funzioni hash utilizzate.

### 2.3.2 - Virtual server

In [14] Rao ed altri propongono un approccio radicalmente opposto rispetto alla precedente tecnica: il bilanciamento del carico è effettuato tramite il ridimensionamento dinamico della quantità di spazio degli indirizzi assegnato ad un nodo.

In questa proposta si introduce il concetto di *virtual server* ovvero nodi logici il cui comportamento è indistinguibile da un normale nodo di una rete che non utilizza questa tecnica di bilanciamento del carico. Ad ogni nodo fisico è assegnata la gestione di uno o più nodi logici o virtual server.

In questo modo, è possibile far gestire ad un singolo nodo fisico delle sezioni non contigue dello spazio delle chiavi ai quali è associato un carico di lavoro dato dalle chiavi che contiene. Il carico a cui è sottoposto il nodo fisico è la somma dei carichi di tutti i nodi virtuali che gestisce.

Eseguire Il bilanciamento del carico è dunque un'operazione facilmente realizzabile tramite il trasferimento della gestione di un virtual server da un nodo fisico carico ad un nodo fisico scarico. Questa operazione è implementabile come un'operazione di leave seguita da una join, entrambe alla base di ogni DHT, per cui sono questo sistema di bilanciamento del carico è facilmente utilizzabile in tutte le DHT.

Nell'articolo sono descritte le regole da seguire per effettuare dei trasferimenti che riducano lo sbilanciamento globale del carico:

1) I nodi, in base al loro carico di lavoro, si suddividono in due categorie: nodi scarichi o *light node*, e nodi carichi o *heavy node*. I nodi si assegnano uno di questi stati confrontando il proprio carico con un valore di soglia.

2) Solo gli heavy node possono cedere un virtual server.

3) Il virtual node trasferito è il più piccolo che rende l'heavy node un light node.

4) Il trasferimento del virtual server non renderà il nodo fisico ricevente un heavy node.

5) Se non esiste nessun virtual server il cui trasferimento rende l'heavy node un light node, allora si trasferisce il virtual server più carico.

Per effettuare lo scambio di virtual server è comunque necessario che gli heavy node conoscano un light node a cui trasferire il carico in eccesso.

Nell'articolo sono proposti due algoritmi per l'individuazione dei light node: il primo che mette in comunicazione un heavy node con un unico light node, mentre il secondo permette all'heavy node di scegliere il nodo fisico più scarico fra un insieme di light node.

Il primo algoritmo è di semplice realizzazione: periodicamente un light node sceglie un ID casuale ed interroga il nodo relativo, se questo è un heavy node allora richiede il trasferimento di un virtual server.

Il secondo algoritmo utilizza invece delle “directory” nelle quali i light node si registrano. Un heavy node può dunque trovare l'elenco dei light node semplicemente interrogando la directory da cui sceglierà il nodo che permette di renderlo un light node con il minor trasferimento di dati possibile.

*A* questa tecnica di bilanciamento del carico può essere avanzata la critica di incrementare la lunghezza delle ricerche, in quanto il numero dei nodi virtuali che partecipano alla rete è maggiore del numero di nodi reali. C'è però da considerare

che le ricerche sulle DHT crescono in modo logaritmico rispetto il numero dei nodi, per cui l'impatto di questo incremento è piuttosto limitato.

Uno svantaggio intrinseco di questa tecnica è dovuta all'incremento dell'utilizzo di risorse come la banda che avviene durante l'operazione del bilanciamento del carico. Infatti per questa operazione si esegue un trasferimento di dati fra due nodi fisici differenti, di cui uno è già sovraccarico. Inoltre va considerato il sottosistema di replicazione delle chiavi, implemento in tutte le DHT, che è utilizzato per evitare la scomparsa dalla rete di chiavi gestite da nodi che abbandonano la retea causa di fallimenti improvvisi. Il mantenimento della replicazione delle chiavi richiede il trasferimento di dati sia durante l'operazione di join che durante l'operazione di leave. Nel primo caso infatti il nuovo nodo oltre ad acquisire le chiavi ad esso assegnate copia anche le chiavi di alcuni suoi vicini, nel secondo caso i nodi rimasti nella rete devono ripristinare la ridondanza perduta a causa dell'abbandono del nodo dalla rete.

Un altro problema è dovuto alla maggiore sensibilità della rete ai guasti: un nodo fisico che subisce un guasto causa l'abbandono improvviso dalla rete di tutti i virtual server che gestisce. I parametri che garantiscono la resistenza ai guasti dovranno essere rivisti, in quanto più nodi logici saranno influenzati dal fallimento di un singolo nodo fisico.

### 2.3.3 - Analisi asintotica delle tecniche di inserimento casuale dei nodi

L'efficienza del consistent hashing è molto sensibile alla selezione del punto di ripartizione dello spazio degli identificatori durante l'operazione di join di un nuovo nodo.

In [11] sono mostrati i risultati di uno studio che analizza diverse varianti di tecniche di inserimento casuale dei nodi all'interno della rete, confrontandone le metodologie per trovare quali di queste offrono un minore sbilanciamento della

dimensione dello spazio degli identificatori assegnato ai nodi, e quindi, di conseguenza il numero di chiavi ad essi assegnati.

Nel documento sono trattate le seguenti coppie di caratteristiche contrapposte:

1. inserimento del nodo con campionamento del punto di inserimento singolo o multiplo
2. campionamento del punto di inserimento con tecniche totalmente casuali o parzialmente deterministiche
3. divisione dello spazio delle chiavi casuale o centrale.

La prima coppia di questa analisi, analizza quali vantaggi si ottengono interrogando un numero di nodi maggiore di uno durante l'operazione di join. L'incremento del numero di nodi contattati durante questa operazione permette di selezionare, come punto di inserimento del nuovo nodo, quello che ricade nello spazio degli identificatori più ampio e quindi che potenzialmente gestisce il numero più elevato di chiavi.

Come metro di valutazione di questa analisi è stato utilizzato il valore assunto da $f_{max}$ ossia il rapporto fra la dimensione della zona dello spazio degli identificatori più ampia presente nella rete e la dimensione della media dello spazio degli identificatori assegnato ai nodi. Ovviamente il valore ottimo di $f_{max}$ si ottiene quando ogni nodo che si inserisce nella rete analizza la dimensione dello spazio degli identificatori di ogni nodo. Questa operazione però farebbe lievitare il costo dell'operazione di join da $O(\log(n))$ a $O(n)$, riducendo notevolmente la scalabilità di una DHT che utilizzasse questa tecnica. Per cui lo scopo di questa analisi è quella di studiare l'andamento della riduzione di $f_{max}$.

I risultati ottenuti con questa analisi sono mostrati dall'Illustrazione 7: il valore di $f_{max}$ decresce asintoticamente all'aumentare del numero di punti esaminati durante l'inserimento.

Date le evidenti migliorie mostrate anche in caso di un basso numero di campioni, hanno proseguito l'analisi cercando di ridurre il traffico di rete generato

per interrogare questi punti. Infatti con una selezione completamente casuale di questi $d$ punti, generata per esempio applicando un egual numero di funzioni di hash al IP del nodo, si ottiene che il numero totale di messaggi scambiati fra i nodi sia $O(d*\log(n))$ .

L'approccio seguito è stato quello di scegliere in modo pseudocasuale soltanto un numero limitato punti, e partendo da questi scegliere gli altri punti tramite un algoritmo predefinito. In particolare hanno studiato il caso in cui il primo punto viene selezionato tramite l'utilizzo di una funzione hash, mentre i restanti $d-1$ punti sono determinati utilizzando i collegamenti del nodo selezionato. Il vantaggi in termini di messaggi scambiati è evidente: si elimina la necessità di eseguire $d-1$ operazioni di ricerca riducendo così il numero di messaggi scambiati da $O(d*\log(n))$ a $O(\log(n)+(d-1))$ .

La riduzione di casualità nella selezione dei punti di inserimento, influisce negativamente sull'efficienza dei risultati: nell'Illustrazione 7 si evidenzia come la tecnica completamente random(colonnina bianca) ottiene degli $f_{max}$ di dimensione minore rispetto alla tecnica parzialmente deterministica (colonnina nera).

Un'ulteriore analisi effettuata in [11] riguarda la tecnica di ripartizione dello spazio degli identificatori, cioè la determinazione di quale parte dello spazio delle chiavi dovrà essere assegnata al nodo che sta entrando a far parte della DHT.

Questo studio affronta il problema dell'assegnamento di zone di dimensione esigue dello spazio delle identificatori che provoca la presenza nella rete di nodi che contengono pochissime chiavi.

Con questa analisi, il risultato della funzione di hash applicata all'IP del nodo non genera più l'identificatore che sarà utilizzato dal peer ma soltanto un identificatore che permette di selezionare il nodo che dovrà cedere parte della propria sezione dello spazio degli identificatori: l'identificatore del nuovo peer è assegnato dal nodo che cede lo spazio degli indirizzi.

In questo modo è stato possibile studiare come varia il valore medio di $f_{max}$ al variare del punto di ripartizione dello spazio degli identificatori: i risultati sono

mostrati nell'Illustrazione 8. In questa figura oltre ad $f_{max}$ , mostra anche $f_{min}$ , ovvero il rapporto fra la dimensione della sezione media e la dimensione della sezione più piccola.

Da questa analisi è risultato che sia $f_{max}$ che $f_{min}$ sono fortemente legati alla tecnica di selezione del punto di ripartizione dello spazio degli identificatori, ed entrambi mostrano un punto di ottimo quando lo spazio degli identificatori è suddiviso a metà fra il nodo che cede lo spazio ed il nuovo nodo che entra a far parte della rete.

## 2.4 - Sistemi che non utilizzano il consistent hashing

Il consistent hashing, con i dovuti accorgimenti, svolge un buon lavoro nelle operazioni di bilanciamento del numero di chiavi assegnate ad ogni peer. Come è stato dimostrato in [13] il numero di chiavi ricevute dal nodo più carico è superiore di un fattore logaritmico rispetto alla media.

Per ottenere un tale risultato distribuisce in modo pseudo casuale le chiavi nello spazio degli indirizzi indipendentemente dai valori associati alle chiavi, per cui è probabile che degli identificatori molto differenti vengano assegnati a due chiavi molto simili tra di loro: in pratica è del tutto eliminata la località dei dati.

Sebbene l'assenza di località non sia un problema nell'utilizzo tipico per cui sono state progettate le DHT, ovvero la registrazione ed il recupero del valore associato ad un'unica chiave, ciò rende molto difficoltoso estendere le funzionalità di questi sistemi P2P.

La possibilità di eseguire query complesse, come range query, ovvero la ricerca di tutte le chiavi i cui valori sono compresi fra due estremi, o query che si basano sull'analisi del contenuto di una chiave come per esempio tutte le chiavi i cui valori contengono una data stringa, sono caratteristiche molto desiderabili ma di difficile realizzazione.

L'utilità delle query complesse è facilmente immaginabile: la possibilità di effettuare questo tipo di query permetterebbe alle reti P2P di essere applicate in nuovi ambiti offrendo un sistema affidabile, scalabile e completamente decentralizzato. Per esempio si potrebbero utilizzare come Grid information services, ovvero un servizio delle griglie che permette la localizzazione delle risorse disponibili in una griglia.

Per questo motivo molti sforzi sono stati compiuti per estendere le funzionalità delle DHT ed includere la capacità di eseguire query complesse sulle chiavi presenti nella rete. Molte tecniche proposte si basano sull'introduzione di un indice logico intermedio, che modificala funzione di distribuzione delle chiavi con

la conseguenza di eliminare o comunque ridurre l'effetto di distribuzione omogenea delle chiavi nello spazio degli identificatori [16] [17].

Un effetto collaterale di questi sistemi è una riduzione di efficienza del consistent hashing del suo compito di bilanciamento del carico sui nodi della rete. Un esempio di un tale sistema è Squid.

### 2.4.1 - Squid

Squid [17] è un sistema che permette di eseguire delle query complesse su una DHT, sostituendo l'utilizzo del consistent hashing per l'assegnazione dell'ID ai dati con un sistema che permette di mantenere la località: due oggetti mappati nel sistema con valori simili avranno buona probabilità di avere un ID vicino. Squid per assegnare l'ID agli oggetti immessi nella rete utilizza un sistema complesso che si basa sugli attributi stessi dei dati da inserire.

Un oggetto che viene inserito nel sistema è caratterizzato da un insieme di attributi per il quale può essere ricercato: per esempio potremmo richiedere che in un sistema che gestisce un insieme di computer vengano selezionati per la ricerca la quantità di RAM ed il tipo di CPU. Supponendo di utilizzare oggetti con al massimo *n* attributi diversi di cui si desidera indicizzare il valore, si crea uno spazio *n*-dimensionale, a cui per ogni dimensione corrisponde un attributo. Ogni oggetto sarà dunque posizionato nello spazio n-dimensionale in base al valore dei suoi attributi ed in base a questo gli verrà assegnato un ID che permetterà di inserire l'oggetto in una DHT come ad esempio Chord.

La posizione dell'oggetto nello spazio non è però utilizzabile come ID, per cui è necessario utilizzare una funzione che “linearizzi” lo spazio n-dimensionale.

Per far ciò si utilizzano le space filling curve, delle funzioni continue $f : N^d \rightarrow N$ che mappano ogni punto di uno spazio ad n dimensioni in uno spazio ad un'unica dimensione. Questa operazione è eseguita mediante un'approssimazione, effettuata tramite una discretizzazione dello spazio in cubi di n dimensioni la cui dimensione dipende dal livello di approssimazione desiderato.

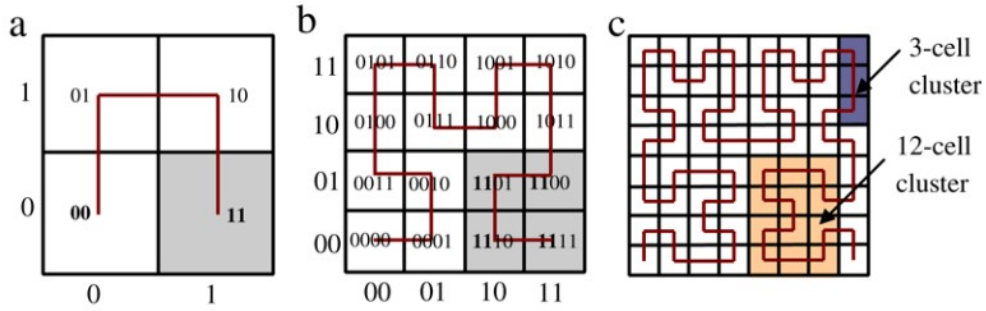


*Illustrazione 6: Approssimazioni successive della space filling curve di hilbert. Nell'immagine b è anche mostrata la mappatura del valore di ogni singolo "cubo" a 2 dimensioni. Nell'imagine c si vedono due esempi di cluster.*

Ad ogni cubo n-dimensionale è assegnato un valore che verrà utilizzato come ID per tutti gli oggetti che ricadono in quel cubo.

L'utilizzo di una space filling curve come quella di Hilbert permette di mantenere la località dei dati: due cubi vicini avranno buone probabilità di essere mappati in zone vicine della linea. Per questo motivo, gli oggetti che ricadono in due cubi vicini avranno buone probabilità di essere mappati su due nodi vicini della DHT.

Eseguire una range query in un tale sistema consiste nel selezionare un sotto insieme di cubi limitrofi che dovranno poi essere ricercati nella DHT. I blocchi ottenuti saranno divisi in un numero di cluster, ovvero un insieme di blocchi limitrofi dopo la linearizzazione effettuata con la space filling curve. Per ognuno di questi cluster sarà necessario effettuare una ricerca sulla DHT per individuare i nodi sui quali sono stati mappati gli oggetti ricercati.

Si può subito notare come il bilanciamento del carico di un tale sistema sia critico: il bilanciamento del carico offerto dal consistent hashing è stato sacrificato per mantenere la località dei dati. Se i valori degli oggetti sono tali da ricadere in un numero ridotto di cluster, allora pochi nodi dovranno gestire una quantità di carico molto elevata.

Per alleviare gli effetti di sovraccarico dei nodi non è possibile intervenire sul posizionamento delle chiavi sui nodi, poiché su di esso è basato l'intero sistema di ricerca. È però possibile intervenire sul posizionamento dei nodi: nell'articolo si

propone l'utilizzo di due possibili tecniche, una dei virtual server già presentata in questo capitolo, l'altra basata sulla richiesta del livello di carico di un numero di nodi prima di scegliere il punto in cui inserire un nuovo nodo.

Nell'articolo non sono però mostrati risultati riguardanti il carico a cui i nodi sono sottoposti, né come esso venga ripartito fra i nodi.

## 2.5 - Conclusioni

Il bilanciamento del carico è una componente fondamentale dei sistemi P2P, senza di esso è impossibile realizzare sistemi scalabili con buoni tempi di risposta. Il consistent hashing è stato fondamentale per lo sviluppo delle DHT, garantendo la loro tipica scalabilità ed efficienza, ma si è dimostrato molto complesso da modificare per estendere le funzionalità di queste reti.

La completa distruzione della località dei dati garantisce da un lato un buon bilanciamento del carico, dall'altro è stato il principale freno all'estensione delle funzionalità delle DHT che tentano di includere un sistema di query complesse.

Le reti DHT più recenti come per esempio Squid hanno accettato il compromesso di una ridotta efficienza del bilanciamento del carico sui nodi al fine di poter preservare parte della località dei dati.

Si sono dunque aperte molte possibilità di ricerca riguardanti il bilanciamento del carico su queste nuove reti. Il lavoro che presento avanza una proposta su una tecnica per l'individuazione dei nodi sovraccarichi della rete basata sull'analisi dinamica del traffico che attraversa i singoli peer.

Conoscendo la posizione dei punti sovraccarichi della rete è possibile, tramite una modifica dell'operazione di join, inserire i nodi lì dove il carico è maggiore di un certo livello di soglia, ottenendo così un sistema di bilanciamento del carico.

# Capitolo 3 - Rilevazione di hot spot nelle Distributed Hash Table

In una rete peer to peer la quantità di lavoro che deve essere ripartita fra i nodi che la costituiscono è la somma del traffico delle richieste effettuate dagli stessi nodi che vi partecipano. Ognuno dei peer dovrà gestire una parte del traffico totale della rete mettendo a disposizione parte delle sue risorse hardware. La metodologia con la quale si effettua questa suddivisione determina il grado di bilanciamento del carico fra i nodi.

Una corretta suddivisione del carico è una caratteristica molto desiderata in una rete P2P in quanto permette all'intero sistema di fornire migliori prestazioni: in caso di sbilanciamento del carico, ad alcuni nodi può essere assegnato un numero di richieste molto elevato, che se eccede le loro capacità di gestione, causa un incremento dei tempi di servizio o addirittura l'impossibilità di gestire alcune richieste; mentre gli altri nodi si troverebbero a gestire quantità di lavoro limitate che non sfruttano pienamente le risorse messe a disposizione dai nodi.

## 3.1 - Definizione di carico di lavoro

Fino ad ora ho utilizzato il termine “carico di lavoro” con una connotazione molto generale, ovvero come la quantità di risorse necessarie per gestire le chiavi assegnate ad un nodo.

Questa definizione fornisce soltanto un'idea generica che non permette di quantificare il carico di lavoro assegnato ad un nodo. Per poter confrontare il carico assegnato a due diversi peer è dunque necessaria una definizione più rigorosa che permetta di definire una “unità di misura” del carico.

Nel consistent hashing descritto nel precedente capitolo, il concetto di carico di lavoro assegnato ad un nodo è definito come:

### definizione 1:

*il carico di lavoro di un nodo è proporzionale al numero di chiavi ad esso assegnate.*

Sebbene questa definizione permetta di definire un'unità di misura del carico di lavoro facilmente rilevabile, essa è stata poco utilizzata dai sistemi di bilanciamento che le preferiscono un'altra definizione che si basa sulla dimensione dello spazio degli identificatori assegnato al nodo.

Infatti i sistemi di bilanciamento del carico mostrati nel capitolo precedente si basano sulla seguente definizione:

### definizione 2:

*Il carico di lavoro associato ad un nodo è proporzionale alla dimensione dello spazio degli identificatori ad esso assegnato*.

Quest'ultima definizione è più “dinamica” in quanto è legata alla probabilità che ha un nodo di ricevere nuove chiavi, e quindi può in qualche modo seguire l'andamento del carico sul nodo.

Un sistema di bilanciamento del carico che si basa sulle definizioni 1 e 2, agisce sotto l'ipotesi che il carico di lavoro associato ad una chiave sia poco

variabile rispetto alla media, o che esistano i presupposti statistici per cui la somma dei carichi associati alle $n$ chiavi assegnate ad un nodo non si discosti considerevolmente da $n$ volte il carico medio di una chiave.

Infatti queste definizioni si basano direttamente o indirettamente sul bilanciamento del numero di chiavi associate ad un peer, per cui se l'ipotesi precedente non è valida allora due nodi con un numero di chiavi identico potrebbero gestire quantità di carico molto differenti.

La seconda definizione di carico di lavoro si basa anche su un'altra assunzione implicita: essa è tanto migliore, quanto la distribuzione delle chiavi è uniforme. Le implementazioni del consistent hashing utilizzate sia in Chord che in Pastry sono un esempio in cui questa assunzione è rispettata: al costo dell'annullamento della località dei dati si ottiene una distribuzione pseudocasuale ed uniforme dei dati sull'intero spazio degli identificatori.

Purtroppo la necessità di estendere le funzionalità delle DHT introducendo query complesse, ha richiesto di sacrificare la distribuzione uniforme fornita dal consistent hashing: la funzione di hash è spesso sostituita con una "locality preserving hash function". Questo tipo di funzioni di hash hanno la caratteristica di mantenere, anche solo parzialmente, la località delle chiavi assegnando identificatori "vicini" a due chiavi i cui dati hanno valori simili. Una conseguenza di questo comportamento è l'impossibilità di garantire una distribuzione uniforme sull'intero spazio degli identificatori: la distribuzione dipenderà, anche soltanto parzialmente, dalla distribuzione dei valori associati alle chiavi.

Un'altra considerazione che interessa entrambe le definizioni di carico presentate riguarda la loro "staticità". Consideriamo una rete in cui non vengono inseriti né nuovi nodi né nuove chiavi: in una rete di questo tipo, entrambe definizioni di carico di lavoro mantengono inalterato la stima del carico che associano ad ogni nodo. Analizzando però l'utilizzo delle risorse utilizzate dai peer per la gestione delle chiavi è improbabile che questo rimanga costante, per esempio risorse come la banda di rete utilizzata dipendono fortemente dal numero

di richieste di servizio che un nodo gestisce, per cui molto probabilmente il carico di lavoro dei nodi dovrebbe variare con il tempo.

Per questi motivi, in questo lavoro di tesi, propongo una diversa definizione di carico di lavoro associato ad un nodo:

### definizione 3:

*il carico di lavoro associato ad un nodo per la gestione di una chiave è proporzionale al numero di volte che essa è bersaglio di una query in un lasso di tempo di dimensione prestabilito*.

Da cui deriva che la definizione del carico di lavoro assegnato ad un nodo:

### definizione 4:

*il carico di lavoro associato ad un nodo è proporzionale al numero di query che esso deve gestire in un lasso di tempo di dimensione prestabilita*.

Una caratteristica di questa definizione di carico è di essere in grado di attribuire una quantità di carico differente a due diverse chiavi: ad una chiave poco richiesta sarà associato una quantità di carico minore rispetto ad una chiave oggetto di molte query. È quindi possibile ipotizzare un sistema di bilanciamento del carico che utilizzando questa definizione, attribuisca valori differenti di carico alle due chiavi, rendendo possibile gestirle differentemente.

La definizione 4 è anch'essa soggetta ad alcune limitazioni: essa assume implicitamente che il costo di gestione di una query sia approssimatamente costante, in quanto considera bilanciati due nodi che ricevono il medesimo numero di richieste. Questa ipotesi è comunque poco limitativa poiché la quantità di dati associata ad una chiave difficilmente eccede certe dimensioni. Ciò si riscontra nel fatto che alcune implementazioni delle DHT, ad esempio Bamboo (un'implementazione molto utilizzata di un sistema Pastry), impongono un limite superiore alla quantità di dati che possono essere associati alle chiavi.

La definizione di carico 4 è applicabile anche alle reti P2P di file sharing che notoriamente trattano anche file di dimensione molto elevata. I valori registrati

nella DHT non includono il file in condivisione, ma soltanto i dati relativi al reperimento dei peer che ne mettono a disposizione una copia.

Un sistema di bilanciamento del carico basato su questa definizione deve tenere in considerazione alcuni possibili casi critici: la definizione 4 non tiene in considerazione i costi statici associati alla gestione di una chiave come, ad esempio, la quantità di memoria necessaria a registrare la chiave.

Per esempio ad un nodo potrebbe essere assegnata una quantità di chiavi elevata a cui è associato poco o nessun carico, per cui il carico associato al nodo sarebbe comunque basso. In condizioni estreme, il numero di chiavi potrebbe essere talmente elevato da saturare la capacità di memorizzazione del nodo: il nodo pur essendo considerato scarico, in quanto gestisce un basso numero di query, si troverebbe nell'impossibilità di memorizzare nuove chiavi.

La probabilità di verificarsi del problema appena descritto dipende dalla quantità di memoria richiesta da una chiave per essere registrata e dalla capacità di memorizzazione dei peer. Per cui limitando superiormente la quantità di dati che possono essere associate ad una chiave, come già avviene in molte implementazioni di DHT, oppure incrementando la quantità di spazio di memorizzazione utilizzato dai peer si può ridurre la probabilità con cui si avvera questo fenomeno.

### 3.1.1 - Hot spot e carico di un nodo

Ad ogni chiave che fa parte di una DHT è assegnato un identificatore che ne determina la posizione nello spazio logico a prescindere dal nodo a cui è assegnata. Alcune porzioni dello spazio degli identificatori potranno riceve delle combinazioni “sfavorevoli” di chiavi, ovvero potrà esservi la presenza di un gran numero di chiavi, oppure un numero di chiavi molto richieste, o anche una combinazione delle due. Ciò causa a quella porzione dello spazio degli identificatori di essere “sovraccarica”. Con la seguente definisco il concetto di hot spot**:**

**definizione 5**:

*Un hot spot è una sezione dello spazio degli identificatori che deve gestire una quantità di query tale da renderlo sovraccarico.*

Un nodo è invece sovraccarico se:

**definizione 6:**

*Un nodo è sovraccarico se il numero di query che deve servire è superiore ad un valore di soglia* $\gamma$ .

Si può immediatamente notare come questi due concetti ammettano la presenza di nodi non sovraccarichi all'interno di un hot spot. Infatti il carico di lavoro presente in un hot spot può non sovraccaricare i nodi che lo gestiscono nel caso in cui sia ripartito da un numero sufficiente di peer.

Un nodo sovraccarico al contrario starà gestendo una sezione dello spazio degli identificatori sottoposta ad un numero di query maggiore della soglia $\gamma$ : scegliendo un valore per $\gamma$ sufficientemente elevato, è possibile fare corrispondere ad un nodo sovraccarico la gestione di uno spazio degli identificatori considerato hot spot.

## 3.2 - individuazione degli hot spot

I nodi all'interno dello hot spot saranno dei nodi sovraccarichi se il loro numero non è sufficiente a distribuire il carico in modo tale che ogni peer gestisca un numero di richieste inferiore al valore di soglia $\gamma$ .

Per ridurre il livello di carico dei nodi già assegnati ad un hot spot non bilanciato è possibile incrementare il numero di peer che lo gestiscono: l'inserimento dei nuovi nodi partecipanti alla DHT all'interno dello hot spot permette di alleviare il carico dei nodi già presenti tramite l'acquisizione della gestione di parte dello spazio degli identificatori e le chiavi in esso contenute, bilanciando così il carico di lavoro.

L'introduzione dei nuovi nodi dove il carico non è ben distribuito, è una strategia utilizzabile per costruire un algoritmo di bilanciamento del carico: un sistema così costruito ha il vantaggio di sfruttare la normale ripartizione dello spazio degli identificatori che avviene durante l'operazione di join, quindi non introduce alcun trasferimento aggiuntivo di chiavi fra i nodi rispetto ad una normale DHT.

L'introduzione di un nuovo nodo in un punto sovraccarico necessita di modificare il comportamento dell'operazione di JOIN, essa infatti non prevede alcun meccanismo che permetta di selezionare il punto di inserimento del nodo nella rete: l'identificatore che determina il punto di inserimento nella rete è scelto casualmente dal nuovo peer.

Con un sistema che posiziona il nodo in un punto sovraccarico della rete, questo schema non è più applicabile: il nuovo nodo che sta eseguendo la join non può conoscere lo stato di una rete di cui ancora non fa parte, per cui è necessario che l'identificatore gli sia assegnato dai nodi della rete.

Il punto focale di questa strategia è il sistema di individuazione degli hot spot: in generale le normali informazioni mantenute dai peer non consentono al nodo di conoscere la posizione dei punti sovraccarichi della rete, per cui è necessario

estendere le funzionalità della DHT introducendo strumenti che permettano di stimare il livello di carico delle sezioni dello spazio degli identificatori.

Il primo approccio a cui si può pensare è quello di acquisire informazione sullo stato della rete in modo attivo, cioè interrogando tutti i nodi che ne fanno parte, ed in base ai risultati selezionare il punto più idoneo. Utilizzando questa tecnica i dati ottenuti sarebbero perfetti: il nuovo nodo avrebbe una conoscenza esatta del livello di carico nella rete in ogni suo punto, e potrebbe eseguire l'inserimento nello hot spot più sbilanciato.

Ciò potrebbe essere realizzato tramite un flooding dei nodi, operazione che su una DHT può essere compiuta con la spedizione di $n$ messaggi [20]. Purtroppo ciò comporterebbe un incremento del costo dell'operazione di JOIN da $O(\log(n))$ a $O(n)$, rendendo poco scalabile l'itera DHT.

Per questo motivo un sistema di rilevamento degli hot spot non può richiedere per la sua realizzazione un elevato incremento del numero di messaggi scambiati fra i nodi, ed è preferibile ricorrere a sistemi che acquisiscono informazioni sullo stato della rete in modo passivo.

Dato che il livello di carico di una sezione dello spazio degli identificatori è determinato dal numero di query che la raggiunge, un nodo che fa parte della rete può analizzare le query che lo attraversano al fine di acquisire informazioni sulla distribuzione delle query in modo da poter creare una stima del carico delle sezioni dello spazio degli identificatori.

Una tecnica di individuazione degli hot spot basata su questa osservazione avrebbe il vantaggio di essere totalmente passiva non richiedendo nessuno scambio di messaggi aggiuntivo per la stima del livello di carico dello spazio degli identificatori.

### 3.2.1 - Analisi del routing

Determinare la fattibilità di un sistema di stima del carico che si basi sull'analisi del traffico che attraversa un nodo richiede una conoscenza approfondita sul comportamento dell'operazione di ricerca di un identificatore

nella DHT. L'algoritmo di ricerca di una chiave è simile in tutte le DHT è può essere schematizzato con i seguenti passi:

1. Un nodo sottoposto ad una query, controlla se è il gestore dell'identificatore ricercato, in caso positivo la query termina.
2. Se non è il gestore dell'identificatore, analizza la porzione locale della tabella di routing e seleziona il nodo che minimizza la distanza con l'obiettivo della query.
3. La query è inoltrata al nodo selezionato

Questa operazione è ripetuta mediamente $O(\log(n))$ volte, contattando un egual numero di nodi, prima di arrivare al peer che gestisce l'identificatore ricercato. Ogni nodo coinvolto nell'operazione di ricerca sarà a conoscenza del punto di destinazione della query, e quindi potrà registrare questa informazione in una struttura dati supplementare. Dall'analisi dei dati raccolti in questa struttura è possibile creare una statistica di distribuzione delle query che sono state gestite dal nodo.

Tuttavia non è possibile utilizzare direttamente questi dati per individuare un hot spot: le query che attraversano un nodo non sono un buon campione statistico dell'intera popolazione di query inviate sulla rete DHT. L'insieme di query che attraversa un nodo è fortemente dipendente dalla sua posizione: una condizione necessaria ma non sufficiente per cui un nodo sia coinvolto nel routing di una query è che sia posizionato fra il punto di origine e la destinazione della query.

La distribuzione dei collegamenti della tabella di routing di un nodo non è distribuita uniformemente nello spazio degli identificatori: la frequenza con cui si presenta un link cresce esponenzialmente man mano che diminuisce la distanza fra il nodo e la sezione dello spazio degli identificatori considerata; ciò serve a mantenere una maggiore località nei collegamenti che garantisce alla rete una buona tolleranza ai guasti.

L'elevata località dei collegamenti fra i nodi influenza anche la distribuzione dei messaggi inoltrati: vi è una maggiore probabilità di inoltrare un messaggio verso un nodo vicino rispetto ad un nodo lontano.

Il campione di query che attraversa un nodo la cui destinazione è un peer ad esso vicino sarà più significativo rispetto al campione di query che ha come destinazione un peer lontano, per cui la stima del livello di carico delle zone dello spazio degli identificatori vicini al nodo sarà più affidabile rispetto alla stima delle zone lontane.

Il sistema di individuazione degli hot spot dovrà integrare l'informazione acquisita dai vari nodi della rete aggregando i dati acquisiti dai singoli nodi. Ogni nodo interpellato stima il carico delle zone dello spazio degli identificatori, ed in base a questo seleziona il nodo con le probabilità maggiori di essere posizionato in un hot spot fra tutti i peer con cui è collegato attraverso la finger table.

L'operazione di join di un tale sistema di individuazione degli hot spot è modificata nel modo seguente:

1. Il peer che esegue la join contatta un qualsiasi nodo della rete
2. Il nodo contattato controlla se è un nodo sovraccarico, in tal caso assegna l'identificatore al nuovo peer cedendo parte del suo spazio degli identificatori.
3. Se non è un nodo sovraccarico esamina la sua stima del carico delle regioni limitrofe inoltrando la richiesta di join o in un peer considerato appartenente ad un hot spot oppure, in sua assenza, al primo nodo che non fa parte della regione in cui si è in grado di stimare il livello di carico con la precisione desiderata.

Il risultato dell'applicazione di questo semplice schema, è un routing guidato dalla stima del carico da parte dei nodi, che termina o nel primo nodo sovraccarico che si incontra, oppure dopo aver completato l'intera analisi dello spazio degli identificatori. In quest'ultimo caso l'inserimento del nodo avverrà in modo pseudo casuale.

### 3.2.2 - individuazione degli hot spot : aggregazione delle query

La scelta di una buona struttura dati nella quale registrare le informazioni relative alle query osservate, in seguito indicata come *hit table*, è fondamentale per l'utilizzabilità del sistema di individuazione degli hot spot.

Al fine di non ridurre la scalabilità della rete DHT, la hit table dovrà fornire buoni tempi di risposta permettendo sia una rapida registrazione delle informazioni relative a tutte le query che attraversano un nodo, sia una rapida consultazione della struttura per rispondere rapidamente alle richieste di JOIN ricevute dal nodo.

I nodi di una DHT mantengono soltanto una quantità di dati logaritmica rispetto al numero di peer presenti nella rete per le operazioni di gestione dell'overlay, e poiché lo scopo del sistema di individuazione degli hot spot è di estendere tali capacità di gestione è necessario che la quantità dei dati conservati nella hit table non superi questo limite: l'utilizzo di una forma di aggregazione delle informazioni per compattare i dati mantenuti nella hit table è necessaria per permettere la scalabilità della rete.

Un possibile sistema di aggregazione è costituito dalla suddivisione dello spazio degli identificatori in $k$ parti di uguale dimensione.

Per permettere lo studio della distribuzione dei messaggi inoltrati in base alla distanza dell'obiettivo, gli spazi possono essere numerati considerando la distanza dal nodo: la sezione più vicina avrà indice $i=0$ mentre la più lontana $i=k-1$

.

Alla suddivisione $si_i$ dello spazio degli identificatori corrisponde un casella della hit table $ht_i$ che memorizza il numero di query osservate la cui destinazione ricade in quella suddivisione.

Questa suddivisione, seppure utilizzabile, è innaturale per un nodo di una DHT. I peer raggiungono le diverse zone dello spazio degli indirizzi attraverso i link occupati della routing table, e non hanno alcun modo di spedire un messaggio

ad un nodo la cui posizione è fra due link: per spedire una query ad un nodo in tale posizione devono spedire il messaggio al nodo più vicino alla destinazione di cui son a conoscenza che si occuperà di portare a destinazione il messaggio.

Per questo motivo è stata studiata un'altra forma di aggregazione dei dati, nella quale lo spazio degli identificatori è suddiviso seguendo la posizione dei link della routing table: ad ogni link $rt_i$ del peer corrisponde l'inizio di una nuova suddivisione dello spazio degli identificatori $si_i$ .

La struttura della hit table sarà quindi omogenea alla struttura della tabella di routing del peer: ci sarà una corrispondenza biunivoca fra i link $rt_i$ della routing table del nodo e le entry $ht_i$ della hit table corrispondenti alla *i*-esima sezione dello spazio degli identificatori $si_i$ . In pratica, la hit table può essere vista come un'estensione della routing table che la arricchisce delle informazioni relative al numero di query che attraversano un link.

### 3.2.3 - Individuazione degli hot spot: parametri

Data la corrispondenza fra la hit table e la routing table di un nodo, è possibile considerare sovraccarico il link $rt_i$ della routing table se il numero di messaggi $ht_i$ inoltrati attraverso di esso superi di un certo valore di soglia $\epsilon$ la quantità attesa $\overline{ht_i}$ :

$$(ht_i \geq \overline{ht_i} + \epsilon) \Rightarrow rt_i \, sovraccarico$$

l'algoritmo di individuazione degli hot spot dovrà poter fornire una stima del valore assunto da $\overline{ht_i}$ in ogni momento della vita del peer. Dal valore assegnato a $\epsilon$ invece sarà determinata l'affidabilità delle stime del livello di carico delle sezioni dello spazio degli identificatori: un valore troppo basso provocherà un elevato numero di falsi positivi incrementando il numero di nodi contattati durante l'operazione di join, un valore troppo alto provocherà un elevato numero di falsi negativi che ridurranno la probabilità di individuare un hot spot.

Per poter stimare questi parametri è stato necessario analizzare dettagliatamente il comportamento del routing, operazione eseguita tramite

l'analisi dei risultati di alcune simulazioni preparate in modo da poter estrapolare una "legge" che permetta di calcolare il valore di $\overline{ht_i}$ . Tratterò questo argomento più approfonditamente nel capitolo 4.

Dallo studio effettuato si è ottenuta una curva "non comune" nella quale si distinguono tre sezioni:

1. Il valore relativo al nodo successivo
2. La parte crescente della curva
3. la parte decrescente della curva

la definizione della funzione di fitting è stata eseguita analizzando la curva per parti, in quanto non è stato possibile individuare una curva che approssimasse tutti i punti ricavati dagli esperimenti.

Per stimare se il nodo successore al nodo corrente è in un hot spot, ho utilizzato il confronto diretto fra $ht_0$ ed il valore di soglia $\gamma$ utilizzato dai nodi per determinare il proprio stato di carico.

Questa stima così particolare, è stata utilizzata a causa del forte legame presente fra il numero di query osservate dal nodo verso il suo successore, ed il numero totale di query ricevute dal successore: il valore dell'indice di correlazione è di $0{,}99$ . Un valore così prossimo ad 1 denota una corrispondenza quasi perfetta dell'andamento di questi due valori [21].

Per gli altri punti è stato eseguita un'analisi di regressione non lineare [22] che ha fornito la curva mostrata nel grafico 1 che approssima in modo molto soddisfacente i punti ottenuti dalla simulazione: la somma dei quadrati degli scarti è $s(\beta)=1{,}8902*10^{-05}$ .

La suddetta curva è così definita:

$$\overline{ht_i}=\begin{cases} b*e^{(c*(i-i_{max}))}+d-b & i\geq i_{max} \\ \dfrac{b_1}{1+e^{(b_2-b_3*(i-i_{max}))}} & i<i_{max} \end{cases}$$

dove $\overline{ht_i}$ è il valore calcolato nel punto $i$ , $i_{max}$ è il valore di $i$ per cui si ottiene il massimo relativo della curva, $b, c, d, b_1, b_2, b_3$ sono i parametri della curva di cui sono stati determinati i valori tramite l'analisi di regressione.

La funzione è stata ottenuta dall'analisi dei risultati delle simulazioni con 256 nodi, ed è stata validata con simulazioni contenenti 512 e 1024 nodi. Per reti con un numero maggiore di nodi è necessario estrapolare la funzione: nei casi analizzati, i nuovi punti sono inseriti in corrispondenza di $i_{max}$ .

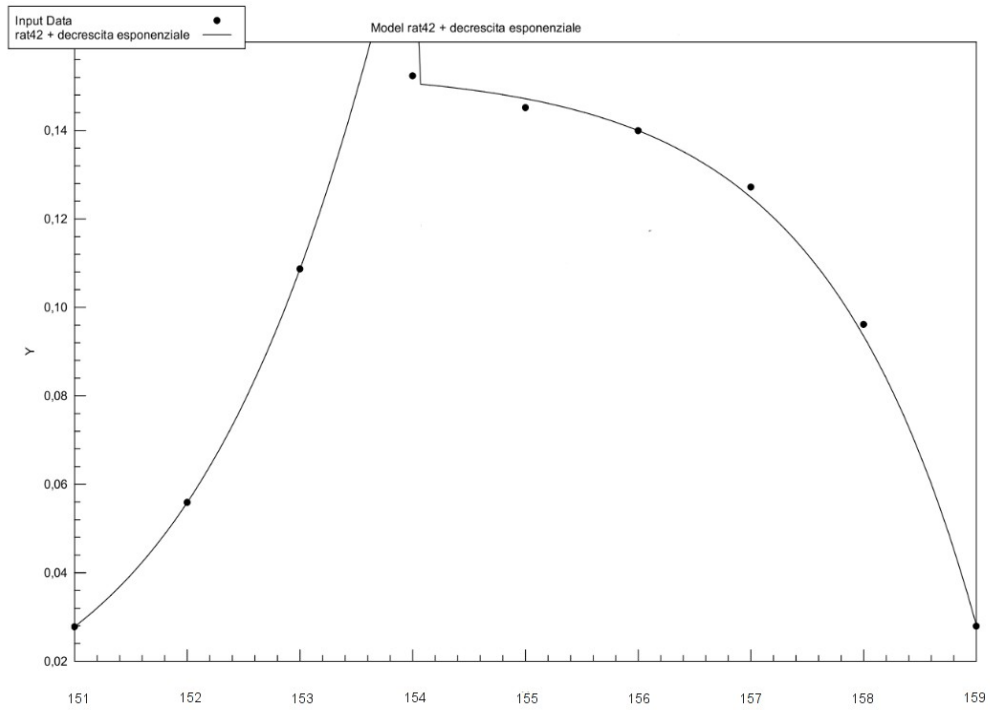


*grafico 1: Valori $ht_i$ ottenuti dalla simulazione e la curva con la quale si calcolano i valori di $\overline{ht_i}$ . Nel grafico il valore di $i_{max}$ è* 154

### 3.2.4 - Determinazione del livello di carico di un nodo

Il carico di lavoro di un nodo dipende dal numero di query che hanno come obiettivo le chiavi da esso gestite: un peer è sovraccarico se in un lasso di tempo di ampiezza prestabilita riceve un numero di query superiore di $\gamma$ . Per poter

determinare univocamente lo stato di carico di un nodo è necessario stabilire una funzione che associ al numero di query ricevute nel tempo dal nodo uno degli stati possibili:scarico o sovraccarico.

Il numero di query che un nodo riceve in una finestra di tempo di dimensione stabilita è un valore che varia nel tempo. Queste variazioni possono essere temporanee oppure strutturali, ossia generate da una variazione o della popolarità o del numero di chiavi gestite. Un esempio di quest'ultimo caso è dato dall'inserimento di un nuovo nodo che acquisendo parte dello spazio degli identificatori allevia il carico del peer.

La funzione di determinazione del carico del nodo, deve essere in grado di distinguere i due casi citati: in una variazione temporanea il nodo deve mantenere il suo stato corrente, mentre con una variazione strutturale deve intervenire tempestivamente modificando il suo stato.

Questa funzione non può utilizzare come valore da analizzare il numero totale di query osservate sin dall'inserimento del nodo nella rete: in tal caso osservazioni molto vecchie temporalmente continuerebbero ad avere un peso significativo nella determinazione dello stato del nodo, generando un sistema poco reattivo alle variazioni del carico.

Limitando la dimensione della finestra temporale di osservazione, ovvero “dimenticando” i dati più vecchi di una certa quantità di tempo, il sistema sarà più reattivo alle variazioni di carico. Con l'utilizzo di un sistema così costruito la funzione di determinazione dello stato del nodo non possiede la capacità di determinare se una variazione del carico è di tipo strutturale o casuale.

Per questo motivo si è deciso di optare per un'altra metodologia basata sull'acquisizione di una serie storica dei dati sulla quale applicare tecniche di analisi statistica che permettono di sgrossare i dati dalle variazioni casuali: in questo modo è possibile dare maggiore risalto al trend di crescita o decrescita seguito dai dati che indica una variazione di tipo strutturale.

La creazione della serie storica è realizzata tramite l'utilizzo di finestre di osservazione di durata prefissata durante le quali si contano il numero di query

che raggiungono il nodo: il numero di query rilevato in ogni finestra di osservazione costituisce un singolo dato della serie storica.

Per poter permettere alla funzione di determinazione dello stato del nodo di seguire l'evoluzione del carico è necessario attribuire un peso ad ogni dato della serie storica che diminuisce man mano che la rilevazione invecchia.

Il sistema descritto è un'applicazione di “double exponential smoothing” [23], ovvero una tecnica di “time series analysis” che permette di analizzare dati storici al fine di effettuare previsioni sull'andamento della serie temporale. Per ogni valore della serie di dati è calcolato il valore di $s_i$ secondo la seguente formula:

$$s_i = \alpha x_i + (1-\alpha)(s_{i-1} + u_{i-1})$$
$$u_i = \theta (s_i - s_{i-1}) - (1-\theta) u_{i-1}$$

dove $x_i$ è il numero di query rilevate nella corrente finestra di osservazione, $s_i$ ed $s_{i-1}$ sono rispettivamente il valore fornito dal double exponential smoothing al tempo corrente ed al tempo precedente, $u_i$ è un valore calcolato che determina la capacità della tecnica di seguire il trend

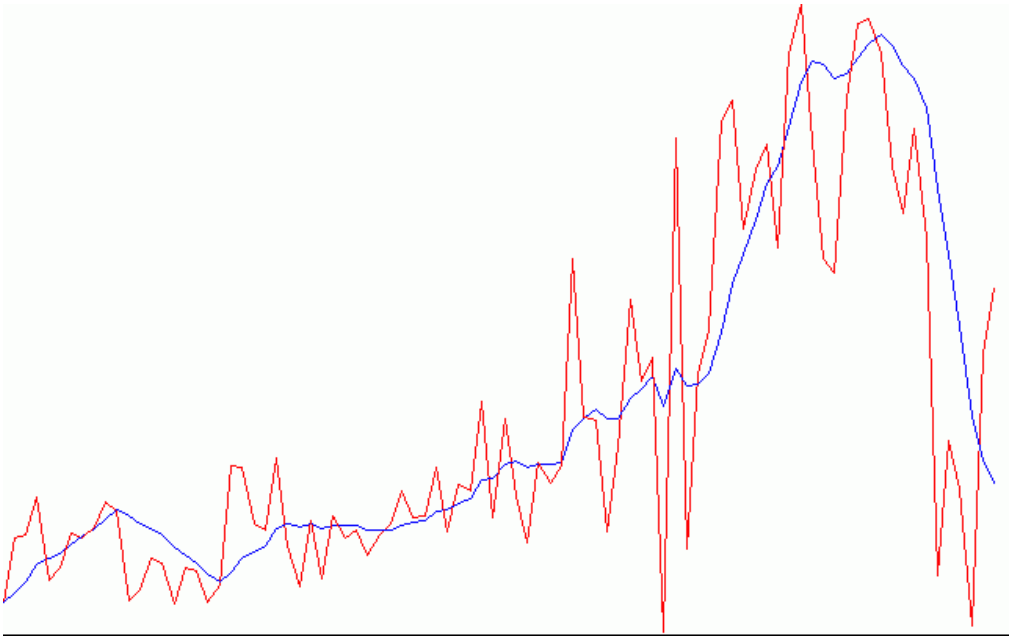

*grafico 2: Esempio di applicazione del double exponential smoothing. La linea rossa raffigura i dati, mentre la linea blu è il valore calcolato: è evidenziata la capacità di tale tecnica di ignorare variazioni casuali anche abbastanza intense, pur riuscendo a seguire il trend dei dati.*

dei dati rilevati, $\alpha$ e $\theta$ sono due parametri che servono a definire il livello di “smoothing” della curva.

La hit table per permettere l'utilizzo di questa metodologia dovrà registrare le informazioni relative ad $x_i$ , $s_i$ , $s_{i-1}$ e $u_i$ , ovvero per ogni sua casella dovrà registrare appena quattro valori numerici.

Lo stato del nodo è quindi determinato dal confronto ,fra $s_i$ e $\gamma$ : se $s_i \geq \gamma$ allora il nodo si attribuirà lo status di sovraccarico, altrimenti si attribuirà lo status di scarico. Questo confronto è ricalcolato al termine della rilevazione dei dati di una finestra di osservazione.

# Capitolo 4 - Analisi del sistema di routing

L'algoritmo di rilevamento degli hot spot effettua una stima del carico di $si_i$ confrontando il numero di messaggi $ht_i$ inoltrati attraverso $rt_i$ con il suo valore "atteso", ovvero un valore medio calcolato $\overline{ht_i}$ : se il valore della rilevazione supera il valore atteso di una quantità sufficiente $\epsilon$ , allora $si_i$ sarà considerato un hot spot.

Questo capitolo mostra gli studi effettuati per determinare le caratteristiche del routing, ovvero l'andamento della distribuzione delle query che attraversano un nodo e l'intensità della variazione della distribuzione in presenza di un hot spot.

Un approccio possibile per determinare le caratteristiche del sistema di routing, è studiare le sue proprietà statisticamente: l'algoritmo di routing è ben definito per cui è possibile studiarne gli effetti analiticamente. Oltre la complessità di una tale operazione vi è un problema ancora maggiore: le applicazioni che implementano i sistemi P2P difficilmente sono completamente conformi alla specifica data. Entrambe le applicazioni utilizzate, Bamboo [24] ed Overlay Weaver [25], hanno mostrato la presenza di alcune differenze fra la loro implementazione e l'algoritmo di routing descritto nell'articolo di Chord [7]. Ciò è dovuto al fatto che nello sviluppo di un'applicazione spesso si cerca di effettuare ottimizzazioni al codice al fine di migliorare le performance. Queste ottimizzazioni pur non modificando la struttura generale dell'algoritmo di routing lo influenzano a sufficienza perché i risultati si discostino da quelli di uno studio teorico.

Per una maggiore accuratezza del lavoro, è stato scelto un approccio orientato all'analisi del sistema di routing effettuato tramite l'esecuzione di simulazioni. Ogni simulazione effettuata ha lo scopo di acquisire dei dati che permettano di estrapolare la "legge" che descrive il sistema di routing: in queste simulazioni, sia il posizionamento dei nodi che l'invio delle query è definito a priori, in modo da

poter ricavare il modello della distribuzione dei messaggi inoltrati attraverso i link della finger table.

Questo capitolo presenta soltanto gli esperimenti eseguiti utilizzando Overlay Weaver configurato in modo da simulare una rete Chord, tralasciando gli esperimenti eseguiti sia con Bamboo che con Overlay Weaver in modalità Pastry. Per le motivazioni rimando al capitolo 5.

Il codice di Overlay Weaver è stato modificato introducendo la hit table e collegandola alla gestione dell'evento che indica il routing di una query: ad ogni messaggio inoltrato viene registrata l'operazione eseguita nella hit table. L'aggiornamento della hit table avviene dopo l'operazione di routing ed è effettuato in modo che non vi siano effetti collaterali che possano influire sul routing di una query.

I paragrafi dal 4.1 al 4.4 presentano i risultati ottenuti da una prima analisi del sistema di routing di overlay weaver. Questo studio ha permesso di perfezionare il modello utilizzato mostrando la correlazione fra gli hot spot e gli incrementi del numero mi messaggi inoltrati attraverso il corrispondente link della finger table.

Da questa analisi è stato anche individuato un bug di overlay weaver che non permette la stabilizzazione della rete Chord.

Per questo motivo è stata eseguita una seconda serie di test, presentati nei paragrafi dal 4.5 al 4.7, i cui risultati sono stati utilizzati per estrapolare la funzione per calcolare $\overline{ht_i}$ .

## 4.1 - Primo esperimento

In questo esperimento, analizzo il comportamento di una rete perfettamente bilanciata in modo da esaminare la distribuzione dei messaggi sui link della finger table in assenza di fattori di sbilanciamento.

Ogni nodo della DHT è posizionato in uno spazio degli identificatori di dimensione $2^{160}$ e gli identificatori assegnati durante l'inizializzazione del nodo vengono perfettamente distribuiti nello spazio degli identificatori: la distanza fra ogni coppia di identificatori successivi è costante.

L'identificatore dell'i-esimo nodo è definita dalla formula:

$$\frac{2^{160}}{numNodi} * i \quad \text{con} \quad i = 0, 1, \ldots, numNodi - 1$$

Ad ogni nodo è affidata la gestione di un'unica chiave con il medesimo ID del nodo.

Il carico di lavoro assegnato ad ogni nodo è anch'esso perfettamente bilanciato: ogni peer gestisce lo stesso numero di query. Esse sono infatti generate in modo che ogni nodo della rete richieda una ed una sola volta tutte le chiavi presenti nella rete.

L'obiettivo di questo esperimento è di fornire una prima indicazione dalla distribuzione delle query sui link della finger table, che sarà utilizzata come termine di paragone per analizzare l'incidenza sulla distribuzione della presenza di una fattore di sbilanciamento del carico.

### 4.1.1 - La preparazione

In questo esperimento ho utilizzato Overlay Weaver configurato in modo da utilizzare l'algoritmo di routing Chord. Il sistema di inoltro dei messaggi scelto è ti

tipo ricorsivo. Per ogni esecuzione dell'esperimento ho preparato uno scenario[1] che svolge le seguenti operazioni:

1. **inserimento dei nodi nella rete**: Sono creati $256$ nodi che eseguono l'operazione di join. L'ID dei nodi è assegnato tramite la formula $\frac{2^{160}}{256} * i$ con $i = 0, 1, ..., 255$ , ma l'ordine di inserimento della rete è casuale in modo da non condizionare la creazione di link stabilendo un ordine preciso. Ogni nodo si inserisce nella rete tramite un altro peer scelto casualmente fra quelli già presenti. Il lasso di tempo che intercorre fra due inserimenti è di 150 millisecondi.

2. **Inserimento delle chiavi**: Ogni nodo esegue l'operazione di PUT di una chiave il cui ID è uguale al proprio identificatore. Durante l'operazione di PUT i nodi non eseguono routing: la chiave è assegnata al nodo che genera la chiave.

3. **Ricerca delle chiavi**: ogni nodo, scelto secondo l'ordine di immissione nella rete, esegue un'operazione di GET verso tutti i nodi della rete. Il totale delle operazioni di GET effettuate dall'intera DHT è uguale al quadrato del numero di nodi presenti.

4. **Mostrare lo stato**: ogni nodo emette in output le informazioni relative alle sue strutture dati, la routing table e la hit table

C'è da notare che a causa della distribuzione uniforme dei 256 nodi sulla spazio degli indirizzi, soltanto le $\log_2(256) = 8$ posizioni più significative della routing table possono mantenere un collegamento con un altro nodo, lasciando tutti gli altri collegamenti inutilizzati: la distanza fra ogni coppia di nodi è infatti $2^{152}$ .

1 In Overlay Weaver uno scenario è un file che descrive un elenco di operazioni che devono essere svolte dai nodi della DHT. È possibile includere tutte le operazioni eseguibili da un nodo (join, put, get...) definendo il momento esatto di esecuzione del comando.

### 4.1.2 - Acquisizione dei dati

L'acquisizione dei dati avviene durante l'operazione di GET, ogni nodo conta il numero di messaggi che ha inoltrato.

Inizialmente ho registrato i messaggi inoltrati nel contatore della hit table corrispondente al link in cui è stato inoltrato il messaggio. L'analisi dei dati acquisiti con questo approccio si è rilevata molto complessa a causa di un'elevata variabilità dei dati, rendendo impossibile estrapolare un modello di distribuzione dei dati.

L'elevata variabilità dei dati è dovuta ad una routing table molto lacunosa: la maggior parte dei nodi (oltre 80%) hanno generato una routing table che manteneva soltanto pochi link con altri nodi. Al posto degli $\log_2(256)=8$ attesi i nodi erano presenti soltanto 2 o 3 link più il collegamento al nodo successivo.

Con un numero così basso di link, il sistema di acquisizione dei dati non è stato in grado di fornire una buona aggregazione dei valori rilevati: attraverso il link $rt_i$ sono inoltrati i messaggi diretti verso $si_i$ più quelli delle sezioni dello spazio degli identificatori successive ad $i$ per i quali il nodo non ha un collegamento migliore di $rt_i$. Per cui il valore rilevato in $ht_i$ non è dipendente esclusivamente dal carico di $si_i$.

Per questo motivo, ho modificato il sistema di aggregazione dei dati utilizzato: in $ht_i$ anziché contare il numero di query che attraversano $rt_i$, sono contenuti il numero di query che hanno come bersaglio la sezione dello spazio degli identificatori $si_i$. In pratica è come inserire i dati nella hit table in assenza di link duplicati nella routing table.

### 4.1.3 - L'analisi dei dati

I dati raccolti in ognuna delle simulazioni di questo esperimento sono stati utilizzati per l'analisi della distribuzione delle query sui vari link dei nodi. I singoli nodi hanno presentato un numero considerevolmente diverso di query inoltrate:

alcuni nodi mostravano un numero di messaggi inoltrati di un ordine di grandezza maggiore della media.

Per poter analizzare la distribuzione è stato necessario annullare gli effetti di questa variazione di carico: il numero di dati che attraversa ogni link è stato considerato utilizzando una percentuale con 4 cifre significative.

In questo modo è stato possibile calcolare il valore medio della distribuzione dei messaggi sui link della tabella di routing, assegnando ad ogni peer il medesimo peso nel calcolo della media.

Nel grafico 3 è mostrata una rappresentazione dei risultati dell'analisi. Sulle ordinate di questo grafico è indicato l'indice della hit table: nel valore 159 sono indicati i messaggi spediti verso $si_{159}$ ovvero verso la sezione dello spazio degli identificatori più lontana e più ampia, mentre con 152 sono indicati i messaggi spediti verso $si_{152}$ ovvero dato il numero di nodi della simulazione verso il nodo successivo.

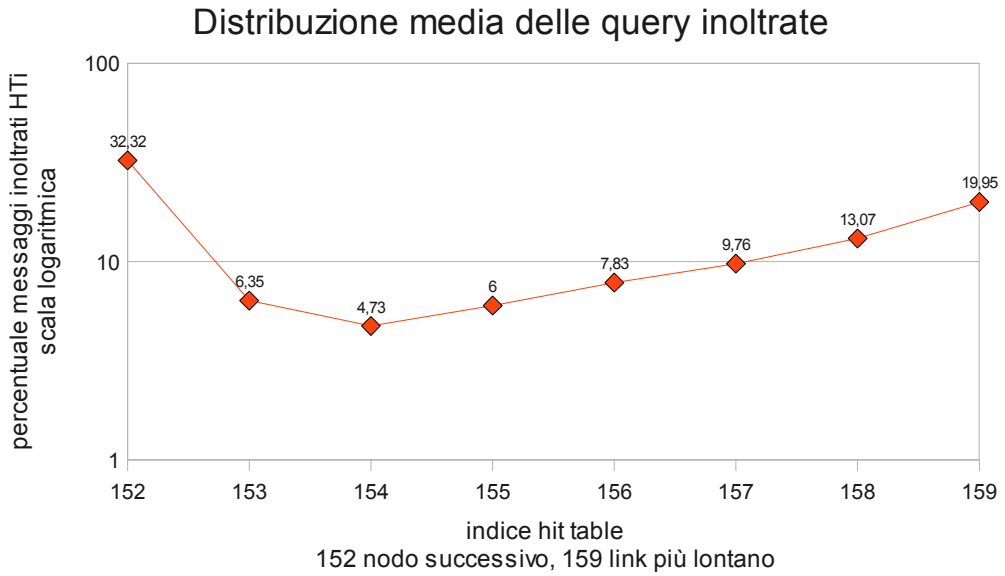


*grafico 3: Distribuzione del traffico inoltrato in scala logaritmica.*

Questo grafico in scala logaritmica mette in evidenza la crescita esponenziale dei messaggi fra la $ht_{154}$ e $ht_{159}$ . Infatti, l'andamento della curva fra quei punti si presenta in una forma molto simile ad una retta.

Si nota inoltre un picco iniziale corrispondente al $ht_{152}$ che corrisponde al collegamento con il nodo successivo.

## 4.2 - Secondo esperimento

In questo secondo esperimento, viene introdotto un hot spot nella rete in posizione nota, in modo da poter analizzare i suoi effetti sulla distribuzione del carico osservata nel precedente esperimento. Lo hot spot è creato aumentando la concentrazione di nodi in una sezione dello spazio degli indirizzi: tutti i nodi continuano a gestire la medesima quantità di carico (sono bilanciati), ma la distribuzione dei nodi incrementa il numero di messaggi ricevuti dalla sezione più popolata dello spazio degli identificatori, rendendolo un hot spot.

### 4.2.1 - La preparazione

Per questo esperimento ho utilizzato Overlay Weaver configurato in modo da utilizzare l'algoritmo di routing Chord. Il sistema di inoltro dei messaggi scelto è di tipo ricorsivo.

Lo scenario che descrive l'esperimento svolge le operazioni descritte nei seguenti punti:

1. **inserimento dei nodi nella rete**: Sono generati 256+k nodi, che ricevono un indirizzo scelto casualmente fra due sottoinsiemi. Il primo sottoinsieme contiene 256 indirizzi che sono disposti equamente nello spazio degli indirizzi, il secondo sottoinsieme contiene $k$ indirizzi disposti in modo da popolare la sezione dello spazio degli indirizzi fra due nodi successivi. Il numero $k$ di nodi inseriti nella sezione più popolata, varia in base all'esperimento da 1 a 64. I nodi per il loro inserimento utilizzano un peer scelto casualmente fra quelli già inseriti nella rete. Il lasso di tempo che intercorre fra due inserimenti è di 200 millisecondi.

2. **inserimento delle chiavi**: su ogni nodo è effettuata un'operazione di PUT di una chiave il cui ID è uguale all'identificatore del nodo. In questo modo la PUT non esegue routing per completare l'inserimento della chiave.

3. **Ricercare le chiavi**: ogni nodo, scelto secondo l'ordine di immissione nella rete, esegue una GET verso tutti i nodi della rete. Il totale delle GET effettuate è $(256+k)^2$ .

4. **Mostrare lo stato**: ogni nodo emette in output le informazioni relative alla routing table ed alla hit table

L'acquisizione dei dati avviene come nell'esperimento precedente: il conteggio è effettuato analizzando la destinazione finale della query.

### 4.2.2 - L'analisi dei dati ottenuti

Questo esperimento è stato replicato con valori variabili di $k$ in modo da analizzare hot spot di consistenza differente. $k$ ha ricevuto il valore di 1, 2, 4, 8, 16, 32, 64 in modo da considerare hot spot abbastanza consistenti. Da notare che il consistent hashing puro fornisce fattori di sbilanciamento di un fattore $\log_2(256+k)\approx 8$ , per cui con la scelta di questi valori per il parametro $k$ ho selezionato dei livelli di carico fino a 8 volte più consistenti o più limitati di quelli normalmente incontrati in una normale rete DHT. Per ogni valore di $k$ sono stati eseguiti 20 diverse simulazioni.

L'analisi della distribuzione dei dati ha richiesto una suddivisione in gruppi dei nodi: il link tramite il quale un nodo raggiunge lo hot spot dipende dalla posizione del nodo rispetto ad esso. Per poter calcolare quindi le medie della distribuzione degli inoltri delle query è stato necessario riunire tutti i nodi collegati al hot spot mediante il link nella medesima posizione nella routing table. Per esempio, il gruppo più numeroso è quello in cui i nodi raggiungono lo hot spot mediante il link $rt_i$ più grande, ovvero $i=160$ . In totale sono stati creati $\lceil \log_2(256+k) \rceil = 9$ gruppi, di cui il meno popoloso è quello con i nodi più vicini allo hot spot e contene un solo elemento. All'aumentare della distanza la popolosità di ogni gruppo raddoppia rispetto a quella del precedente, ad esclusione del gruppo più distante che contiene esattamente i $k$ che sono posizionati nello hot spot.

La capacità di rilevare lo hot spot $si_i$ tramite l'analisi della distribuzione dipende dall'incremento di carico rilevato su $ht_i$ rispetto al suo valore in caso di carico bilanciato mostrato dall'esperimento precedente.

Per poter confrontare i valori di $ht_i$ relativi a due nodi diversi con un numero totale di messaggi inoltrati differente, è stato necessario "normalizzare" il loro carico calcolandolo in modo percentuale. In questo modo è stato possibile ignorare la differente quantità di carico gestita dai nodi attribuendo il medesimo peso ad ognuno di essi nel computo di parametri come la media.

L'incremento della distribuzione nel link che collega il nodo allo hot spot può essere sfruttato per costruire un test che permetta di distinguere i collegamenti a sezioni dello spazio degli identificatori sovraccarichi da sezioni poco cariche: maggiore è l'incremento tanto più efficiente può essere il test, minimizzando sia il numero di errori di prima specie (falsi negativo) che gli errori di seconda specie (falso positivo).

### 4.2.3 - Interpretazione dei grafici

I grafici 4 e 5 mostrano i risultati del secondo esperimento. Sulle ascisse dei due grafici è indicato l'indice della hit table attraverso il quale sono stati ricavati i valori delle varie linee, mentre nelle ordinate è indicata la percentuale del numero $ht_i$ di messaggi inoltrati attraverso il link *i*. Ogni linea del grafico rappresenta un diverso raggruppamento di nodi che hanno in comune l'indice della finger table $rt_i$ tramite il quale inoltrano messaggi al hot spot: il valore corrispondente a questo indice è quello più rilevante per questo studio.

Il nome delle linee del grafico indica l'indice attraverso il quale inoltrano messaggi nello hot spot.

Per esempio nel grafico 4 la linea contrassegnata con 155, mostra i valori relativi al gruppo di nodi che inoltra i messaggi allo hot spot tramite il link $rt_{155}$ : si può notare come il valore di $ht_{155}$ mostri un valore molto elevato per questa curva.

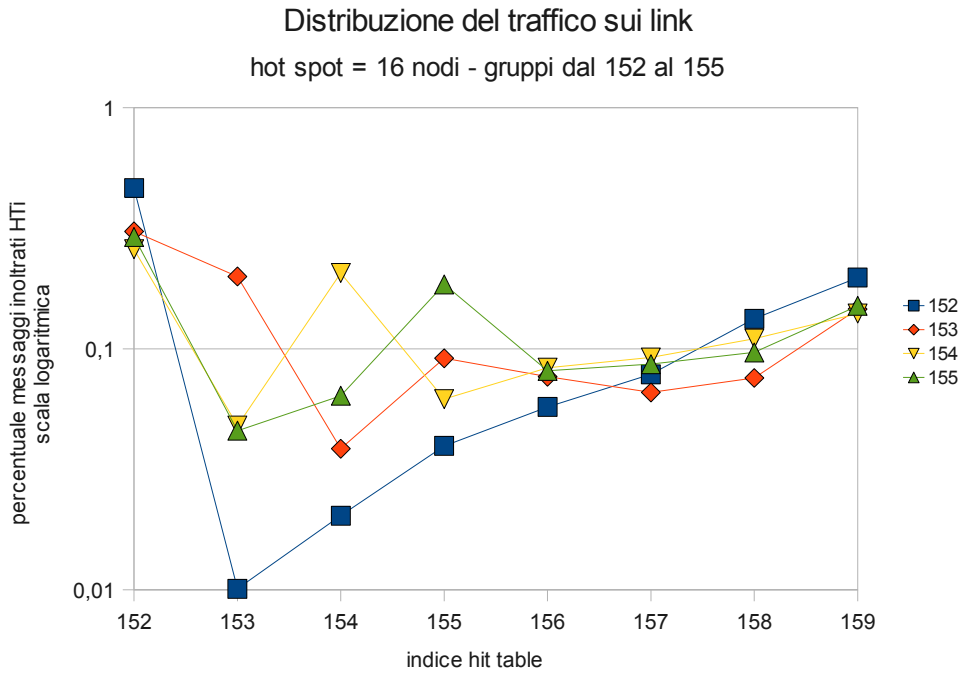


*Grafico 4: Distribuzione delle query sui link della finger table in presenza di hot spot.*

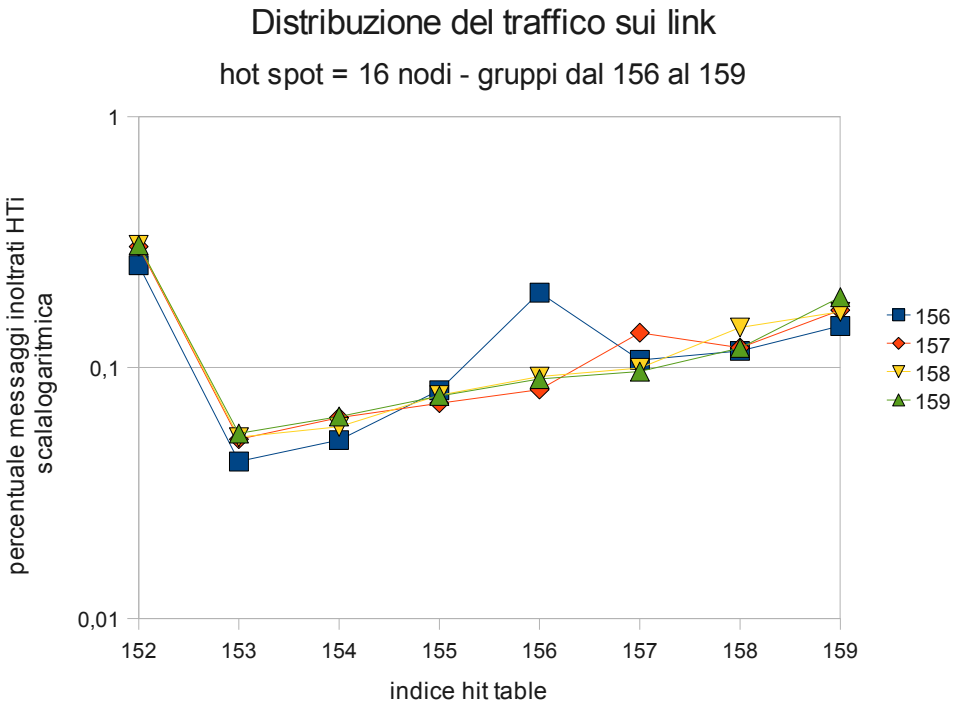


*Grafico 5: Distribuzione delle query sui link della finger table in presenza di hot spot.*

## 4.3 - Determinazione del carico atteso in un link

L'obiettivo degli esperimenti eseguiti è stato quello di fornire un insieme di dati che permettano di stimare [22] il livello di carico di una sezione dello spazio degli identificatori analizzando la distribuzione dell'inoltro delle chiavi effettuate dal nodo. Per far ciò si cerca stabilire se:

$$(ht_i > \overline{ht_i} + \epsilon) \Rightarrow s_i \text{ è un hot spot}$$

ovvero se il numero di messaggi $ht_i$ inoltrati attraverso il link $rt_i$ è maggiore del valore medio atteso $\overline{ht_i}$ più una quantità $\epsilon$ allora la corrispondente sezione dello spazio degli identificatori $s_i$ è da considerarsi un hot spot.

Per eseguire questa analisi è necessario stabilire i parametri $\overline{ht_i}$ ed $\epsilon$ . $\overline{ht_i}$ è in a sua volta dipendente da due parametri: la funzione di distribuzione utilizzata $\phi$ e il link $i$ utilizzato, ovvero $\overline{ht_i} = \phi(i)$ . Per poter eseguire la stima di carico di una sezione dello spazio degli identificatori è necessario stabilire quale sia il valore di $\phi$ .

Da come è stato mostrato dagli esperimenti effettuati, la distribuzione degli inoltri sembrava seguire un andamento esponenziale: per verificare questa ipotesi ho proseguito lo studio cercando di eseguire un fitting dei dati con una curva di questo tipo. Supponendo che la funzione $\phi$ sia un'esponenziale crescente si può scrivere:

$$\phi(i) = c * (1 + t_c)^i$$

dove $c$ è la costante dell'esponenziale ed $t_c$ il tasso di crescita fra due valori consecutivi. Per poter implementare $\phi$ in un algoritmo è necessario valutare il valore dei suoi due parametri.

Il primo approccio seguito è stato quello di determinare entrambi i parametri in modo statico, basandomi sui dati ottenuti dalle simulazioni. Considerando l'elevata variabilità statistica dei dati ottenuti nelle simulazioni, la stima dei parametri di $\phi$ che si può ottenere è molto imprecisa. I singoli nodi hanno

mostrato degli scostamenti significativi sulla forma della curva che pur rimanendo di tipo esponenziale, variava la sua pendenza. La ripidità della curva sembra essere correlata anche all'incremento di messaggi inoltrati verso il link che collega il nodo allo hot spot: nei valori di $ht_i$ corrispondenti ad un hot spot, è presente una variazione del valore tanto maggiore quanto lo è la ripidità della curva.

Per cui ho deciso di abbandonare la determinazione statica dei parametri parametri di $\phi$ in favore di un approccio dinamico eseguito all'interno di ogni nodo.

In questo approccio, i parametri sono determinati mediante l'utilizzo della regressione lineare [26], una tecnica di analisi statistica che permette di eseguire la stima del valore atteso di una variabile dipendente dati i valori della variabile indipendente. Questa tecnica permette di definire la funzione della retta che minimizza il quadrato dello scarto fra il valore calcolato ed il valore dei punti. Per questo motivo è possibile utilizzarla soltanto in distribuzioni lineari o che possono essere ricondotte tramite una trasformazione ad una distribuzione lineare. Nel mio caso di applicazione, la distribuzione utilizzata per modellare i dati è di tipo esponenziale: per poter applicare il metodo della regressione lineare è necessario applicare una trasformazione di tipo logaritmico ai dati.

Il valore calcolato di $\overline{ht_i}$ tramite la regressione lineare utilizzando la trasformazione logaritmica è :

$$\overline{ht_i} = e^{(a+b*i)}$$

con : $a = \overline{T} - b\,\overline{I}$ , $b = \dfrac{cov(I,T)}{var(I)}$ e $t_i = \ln(ht_i)$

dove $\overline{T}$ è la media del vettore dei valori di $t_i$ , che corrispondono alla trasformazione dei valori rilevati $ht_i$ . $\overline{I}$ è la media della variabile indipendente, ovvero gli indici dei link della finger table. $cov(I,T)$ è la covarianza calcolata utilizzando $I$ come variabile indipendente ed il logaritmo di $HT$ come variabile dipendente. $var(I)$ È la varianza della variabile indipendente.

Calcolare la varianza e la covarianza richiede l'esecuzione di alcuni calcoli che riguardano l'intera finger table. Se dovessero essere eseguite molto frequentemente, il carico di lavoro di un nodo in termini di tempo di processore utilizzato potrebbe incrementare significativamente.

Questo problema è però già stato risolto dalla discretizzazione delle acquisizioni dei dati effettuate in finestre di osservazione di lunghezza determinata, descritte nel capitolo precedente. I dati utilizzati per eseguire la stima di $\overline{ht_i}$ non sono i dati grezzi osservati, bensì quelli ottenuti dal sistema di eliminazione della variabilità che ricorre all'utilizzo delle time series analysis. Per cui le operazioni necessarie al calcolo di $a$ e $b$ sono effettuate una sola volta alla conclusione di una finestra di osservazione e rimangono valide fino alla successiva esecuzione del calcolo del double exponential smoothing.

### 4.3.1 - Risultati dell'applicazione della regressione lineare

Ho analizzato l'efficacia del sistema fin qui ipotizzato per la rilevazione degli hot spot applicando la regressione lineare ai dati ottenuti dal secondo esperimento e verificando se i nodi riescono a riconoscere lo hot spot.

La prima operazione è stata quella di calcolare per ogni link di ogni nodo lo scarto fra il valore di $\overline{ht_i}$ , ottenuto tramite l'applicazione della regressione lineare, ed il valore rilevato $ht_i$ . Ho organizzato gli scarti suddividendoli in classi in base alla loro dimensione: ad ogni classe corrisponde il numero di nodi che presentano uno scarto compreso nel range della classe.

I grafici seguenti mostrano il numero di nodi in cui si è presentato un determinato range di scarti. Ogni grafico mostra i dati di un raggruppamento di nodi che condividono l'indice della finger table attraverso il quale sono collegati allo hot spot. Nei grafici sono rappresentati in ascissa le classi di scarto, ed in ordinata il numero di nodi che presentano lo scarto $ht_i - \overline{ht_i}$ il cui valore ricade nella classe di scarto dell'ordinata. In leggenda è indicato il valore dell'indice $i$ dello scarto $ht_i - \overline{ht_i}$ a cui la linea si riferisce.

Analizzando i grafici si nota che:

1. La forma delle linee segue l'andamento della gaussiana, ciò sarà utile per calcolare il valore di $\epsilon$ . I picchi che si notano agli estremi della curva sono dovuti alla scala dei valori delle ascisse molto piccola, per cui le classi che comprendono i valori da $-\infty,-200$ e da $200,+\infty$ ottengono dei valori elevati.

2. La distribuzione della linea che corrisponde al link sovraccarico ha un valore molto spostato sulla destra, ciò significa che per un elevato numero di nodi il link che li collega al hot spot mostra un scarto maggiore rispetto ai link non carichi.

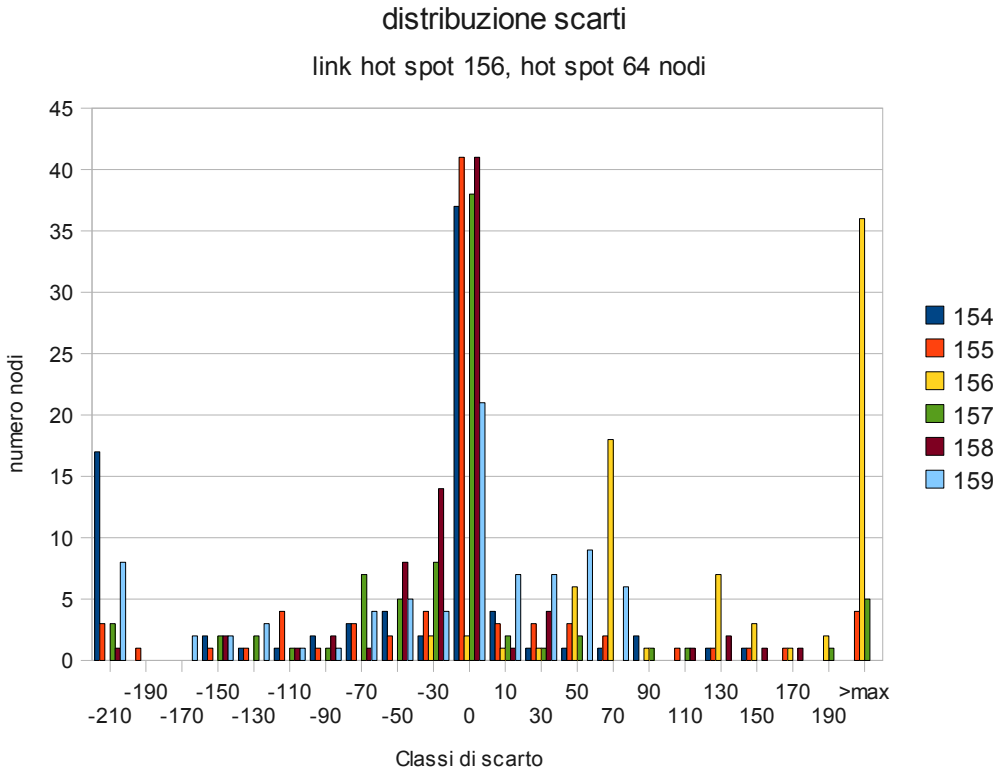


*Grafico 6: Distribuzione degli scarti* $ht_i - \overline{ht_i}$ *: link sovraccarico 156*

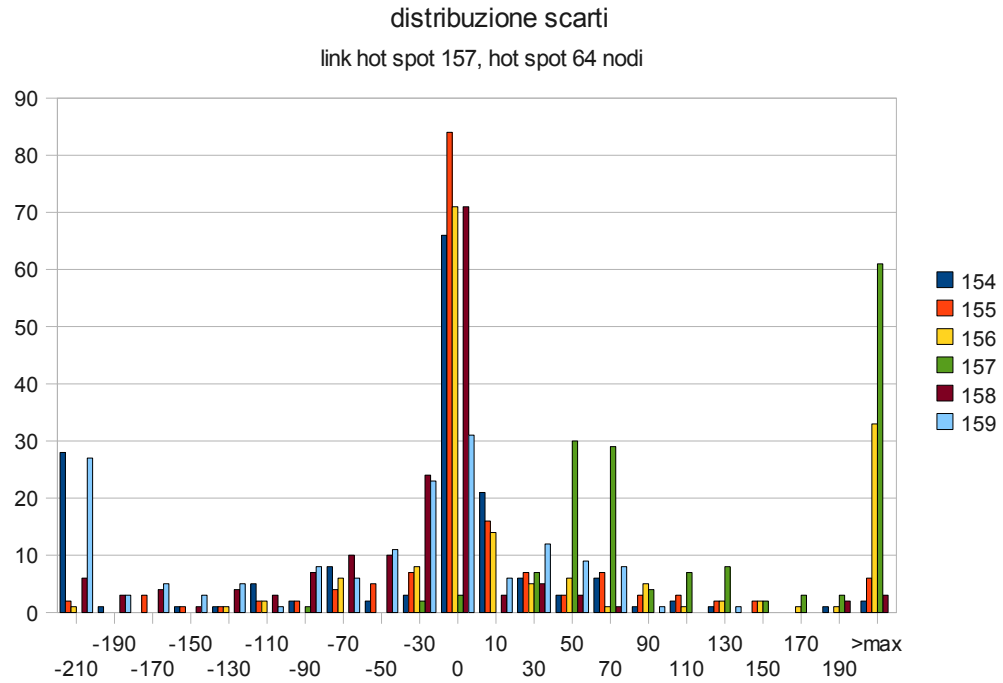


*Grafico 7: Distribuzione degli scarti* $ht_i - \overline{ht_i}$ *. Link sovraccarico 157*

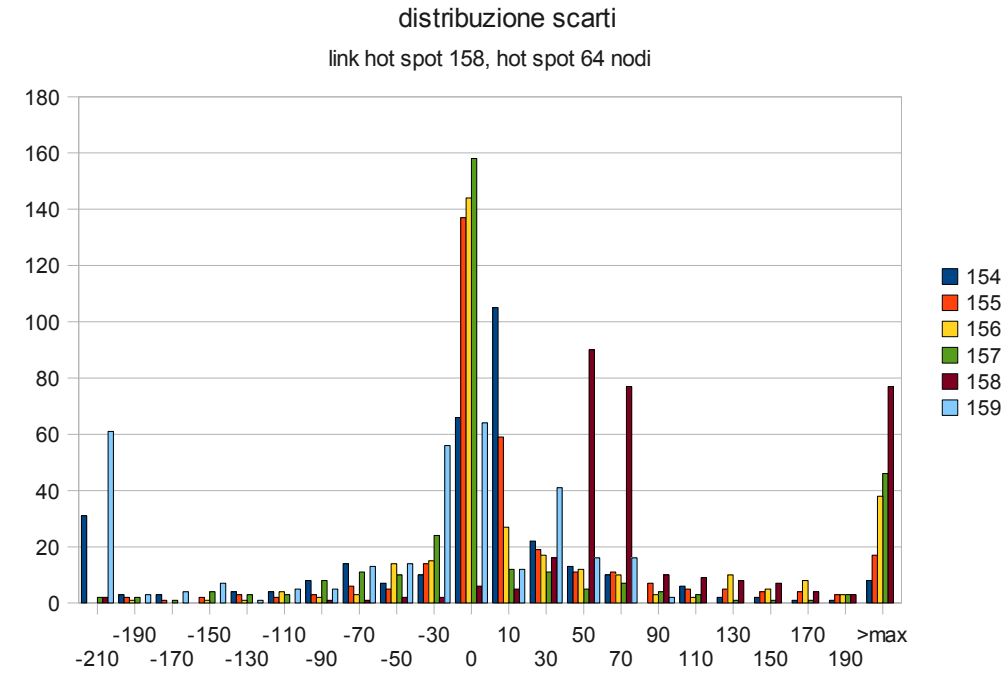


*Grafico 8: Distribuzione degli scarti* $ht_i - \overline{ht_i}$ *. Link sovraccarico 158*

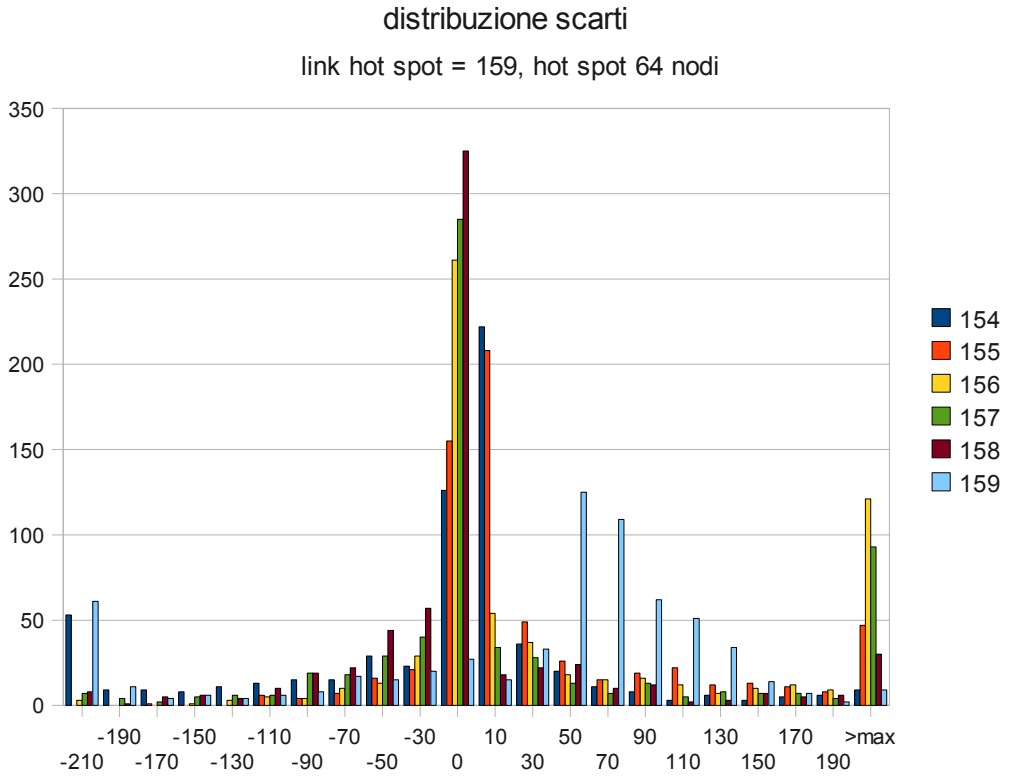


*Grafico 9: : Distribuzione degli scarti* $ht_i - \overline{ht_i}$ *. Link sovraccarico 159*

### 4.3.2 - L'errore ammesso $\epsilon$

Dalla stima di $\epsilon$ dipende l'efficienza del metodo di individuazione degli hot spot. Infatti attribuendogli un valore troppo grande otterremmo una grossa quantità di falsi negativi, mentre con una quantità troppo piccola il problema sarà ribaltato ed otterremo un numero elevato di falsi positivi.

Poiché le curve della distribuzione degli scarti intorno al valore seguono la curva di Gauss, stabilire un valore per $\epsilon$ è piuttosto semplice. Infatti è sufficiente determinare i parametri caratteristici della curva normale (media e varianza) tramite l'analisi di un campione significativo e scegliere la probabilità di commettere un errore: il valore di $\epsilon$ sarà ricavato tramite la consultazione delle tabelle che descrivono la distribuzione normale [22].

### 4.3.3 - Individuazione degli hot spot

Utilizzando i dati che ricavati dagli esperimenti e la loro elaborazione, li ho utilizzati per definire un algoritmo di individuazione degli hot spot. Per far ciò ho creato una simulazione che utilizzasse i dati già raccolti con lo scopo di analizzarne l'efficacia.

Sui dati raccolti da ogni nodo ho calcolato per ogni link della finger table i seguenti valori:

1. il valore di $\overline{ht_i}$ utilizzando la tecnica della regressione lineare.
2. Il valore assoluto dello scarto $ht_i - \overline{ht_i}$
3. Il valore dello scarto relativo dato dal rapporto: $sc1 = \frac{ht_i - \overline{ht_i}}{\overline{ht_i}}$

La sezione dello spazio degli identificatori $si_i$ che è stata stimata dall'algoritmo possibile hot spot è stata quella con il valore massimo di scarto relativo $\frac{ht_i - \overline{ht_i}}{\overline{ht_i}}$ .

Ho anche realizzato un'altra versione di questo algoritmo nella quale lo scarto relativo è calcolato sul valore rilevato, ovvero $sc2 = \frac{ht_i - \overline{ht_i}}{ht_i}$ . I risultati sono confrontati nei seguenti grafici dal 10 al 15.

I grafici mostrano la probabilità di successo dei due algoritmi di inoltrare in un unico hop una richiesta di ricerca di un hot spot attraverso il link corretto. È possibile notare come i nodi presentino una maggiore probabilità di individuare un hot spot ad essi vicino rispetto ad uno lontano.

Questi grafici non mostrano l'efficienza totale del sistema di rilevazione degli hot spot: la struttura di questo sistema utilizza un routing basato sul carico, che aggrega l'informazione raccolta dai diversi nodi per poter stimare con maggiore efficacia la posizione corretta del hot spot.

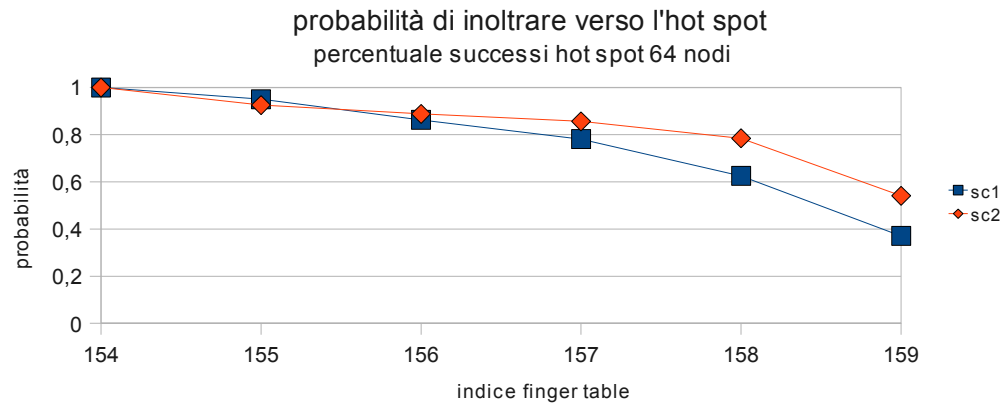


*Grafico 10: Probabilità di un peer di inoltrare un messaggio verso lo hot spot*

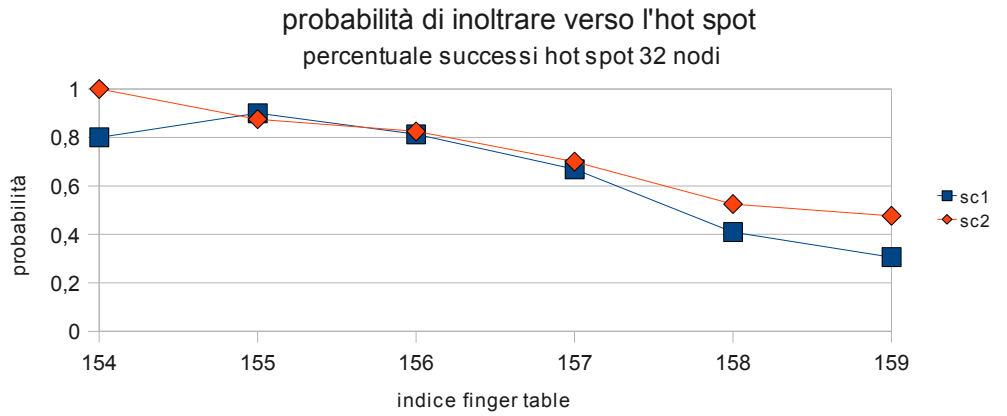


*Grafico 11: Probabilità di un peer di inoltrare un messaggio verso lo hot spot*

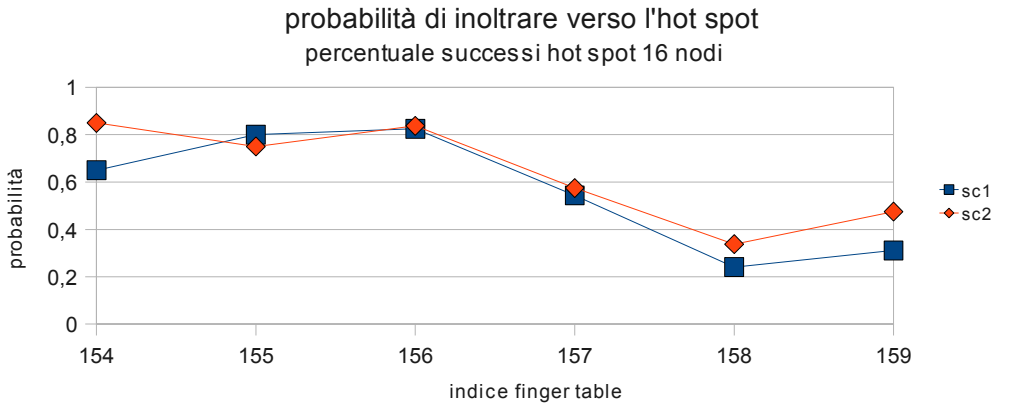


*Grafico 12: Probabilità di un peer di inoltrare un messaggio verso lo hot spot*

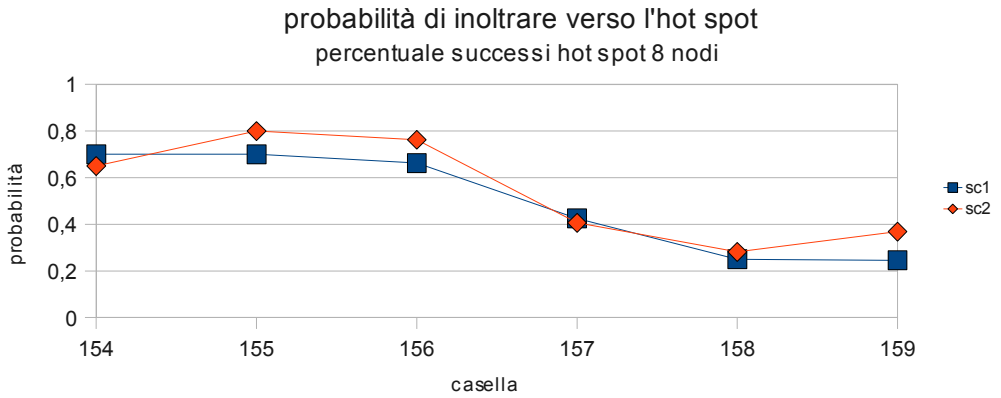


*Grafico 13: Probabilità di un peer di inoltrare un messaggio verso lo hot spot*

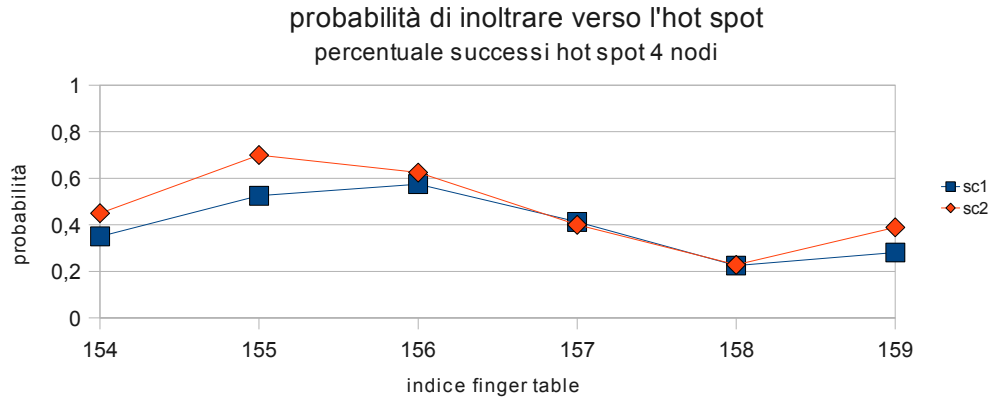


*Grafico 14: Probabilità di un peer di inoltrare un messaggio verso lo hot spot*

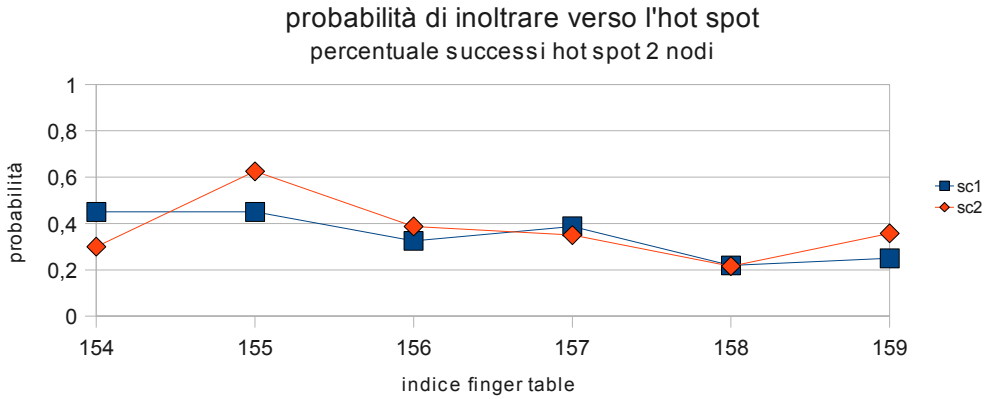


*Grafico 15: Probabilità di un peer di inoltrare un messaggio verso lo hot spot*

Quindi ho eseguito un'altra simulazione, utilizzando i dati fin qui raccolti, che prende in considerazione l'operazione di routing. La probabilità di trovare un hot spot partendo da un nodo qualsiasi, è data dalla probabilità che ha l'operazione di routing di raggiungere lo hot spot. Ho considerato come un insuccesso un'operazione di routing che ha completato un intero giro nello spazio degli identificatori, in pratica è stato considerato come fallimento un'operazione di routing che "superi" la posizione dello hot spot.

Dal grafico 16 si può notare come il sistema presenti delle ottime capacità di trovare un hot spot. In condizioni di sovraccarico simili a quelle riscontrabili normalmente in una rete che utilizzi il consistent hashing, hot spot di dimensione 8, vi è una probabilità di quasi 80% di individuarlo. Questa probabilità cresce all'aumentare della consistenza dello hot spot fino a superare il 95% nel caso con un hot spot di 64 nodi.

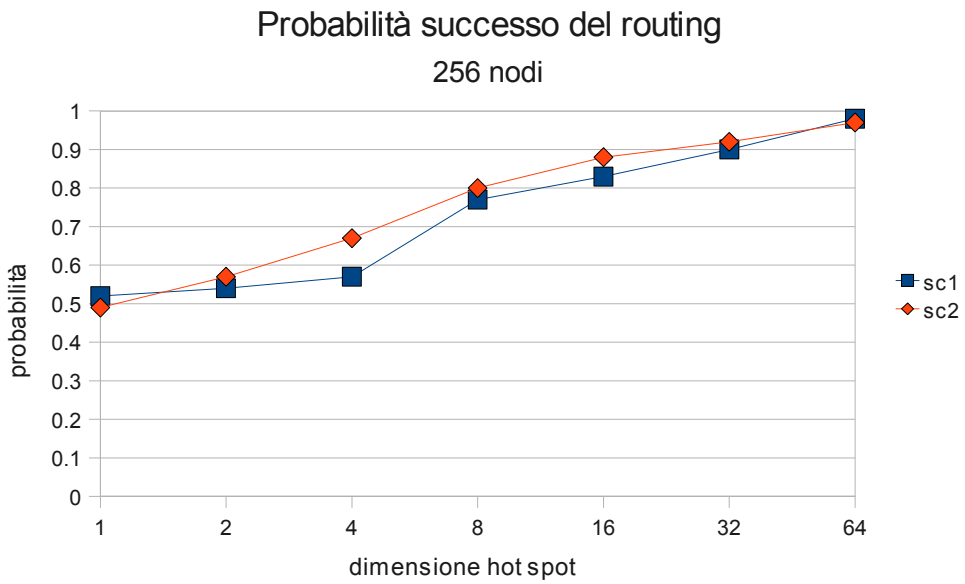


*Grafico 16: Probabilità di individuazione hot spot tramite routing basato sul carico*

## 4.4 - Gli errori presenti negli esperimenti precedenti

L'analisi mostrata fino ad ora dei primi due esperimenti su 256 nodi è stata effettuata contemporaneamente all'esecuzione di simulazioni con la medesima struttura degli esperimenti fin qui illustrati, con reti con un maggiore numero di nodi, 512 e 1024.

La durata di una simulazione è piuttosto lunga: per 1024 nodi è richiesta un'elaborazione continua di circa 22 ore in un PC con buone caratteristiche hardware.

Al termine di un numero di un buon numero di simulazioni, ho verificato se il modello e l'algoritmo sviluppato con i dati raccolti con la simulazione con 256 nodi fosse valido. Il risultato è stato pessimo: la distribuzione delle query sui link seguiva una curva del tutto diversa in quanto non era più identificabile come una esponenziale.

Ho individuato il problema nel basso numero di link “pieni” presenti nella finger table, che non permette di scegliere il migliore nodo per proseguire il routing, influenzando la distribuzione delle query fra i link della routing table. Questo effetto è stato tanto più evidente tanto maggiore fosse il numero di nodi nella rete.

Con un'analisi più approfondita del codice di Overlay Weaver si è messo in evidenza un bug di questo programma: il tempo necessario per la stabilizzazione dei link della routing table, ovvero il tempo di scoperta e di introduzione dei nodi nella finger table, è incredibilmente elevato. Esso dipende da una scelta pseudo casuale, che nel caso più “fortunato” richiede alcune ore affinché gli 8-10 link utilizzati in un nodo vengano stabilizzati. La mancanza di questi link, pur mantenendo un routing di tipo logaritmico grazie alla robustezza della teoria di Chord, causa delle forti influenze sul routing. Di questo problema parlerò più approfonditamente nel capitolo 5.

Inoltre il conteggio dei messaggi inoltrati effettuato da ogni singolo nodo includeva anche i messaggi originati dal nodo stesso: la distribuzione dei nodi e la rilevante percentuale di nodi che inoltrava pochissimi messaggi di altri nodi hanno dato l'andamento esponenziale ai risultati ottenuti.

Per affrontare il problema dei link mancanti ho deciso di intervenire raggirando il bug di Overlay Weaver in modo da garantire l'esecuzione delle simulazioni in una rete in cui tutti i nodi abbiano i link stabilizzati. L'implementazione di Chord su Overlay Weaver è realizzata tramite l'ereditarietà di tre livelli di overlay, ognuno dei quali aggiunge delle funzionalità attese da questa DHT. In ognuno di questi livelli sono presenti delle opzioni di configurazione che permettono di impostare diversi parametri che determinano il funzionamento della DHT. Al livello base (la rete linear walker) vi è un'opzione di configurazione che permette di aggiornare le tabelle di routing durante l'analisi di ogni messaggio ricevuto. Utilizzando questa opzione, alla fine di un esperimento, i nodi presentano nella loro finger table il corretto numero di link utilizzati.

L'aver modificato questa opzione ha influenzato la distribuzione dei messaggi inoltrati attraverso i link della finger table: la curva ottenuta è molto diversa dalla precedente. Ciò è dovuto al fatto che il routing, avendo a disposizione tutti i link possibili, ha migliorato le sue performance: il numero totale di messaggi scambiati nella rete si è ridotto sensibilmente riducendo il numero medio di hop richiesto per completare un'operazione di routing.

La modifica del sistema di routing, dovuta al bug di overlay weaver, ha reso necessario formulare un nuovo modello per la stima di $\overline{ht_i}$ : a questo scopo è stata eseguita una nuova batteria di test per poter stimare le caratteristiche del nuovo sistema di routing nei quali è assicurata l'acquisizione dei dati in una rete stabile.

## 4.5 - Primo esperimento con routing stabilizzato

In questo esperimento analizzo il comportamento del routing esaminando la distribuzione dei dati sui link della finger table in assenza di fattori che possano sbilanciare il carico di lavoro. Rispetto al precedente tentativo di indagine in questa configurazione, le simulazioni si assicurano che la rete Chord sia perfettamente stabilizzata.

I nodi sono posizionati senza l'ausilio della funzione di hash in modo che ognuno abbia associato la medesima quantità di spazio degli identificatori: la distanza fra ogni coppia di nodi successori è costante. La posizione del i-esimo nodo è assegnata tramite la formula:

$$\frac{2^{160}}{n} * i \quad \text{con} \quad i = 0,1,..,n-1$$

Ad ogni nodo è affidata la gestione di una chiave ognuna delle quali ha associato il medesimo carico di lavoro.

In ogni file di scenario sono descritti i seguenti passi:

1. **inserimento dei nodi nella rete**: Sono creati $n$ nodi che eseguono l'operazione di join. L'ID dei nodi è assegnato tramite la formula $\frac{2^{160}}{n} * i$ con $0 \leq i < n$ , ma l'ordine di inserimento della rete è casuale in modo da non condizionare la creazione di link stabilendo un ordine preciso. Ogni nodo si inserisce nella rete tramite un altro peer scelto casualmente fra quelli già presenti. Il lasso di tempo che intercorre fra due inserimenti è di 150 millisecondi.

2. **Inserimento delle chiavi**: Ogni nodo esegue l'operazione di PUT di una chiave il cui ID è uguale al proprio identificatore. Durante l'operazione di PUT i nodi non eseguono routing: la chiave è assegnata al nodo che genera la chiave.

3. **Stabilizzazione della finger table** : questo passaggio assicura che ogni nodo abbia in tutti i link della finger table un collegamento con un nodo della rete. Per fare ciò si esegue su ogni nodo una GET verso tutti i nodi. Questo passaggio è stato sovradimensionato per avere la certezza di una rete Chord stabile.

4. **Azzeramento valori** : Il passo precedente utilizza l'operazione di GET per stabilizzare la tabella di routing. Per evitare che questi valori influiscano sulle analisi, in questo passo si cancellano tutte le hit table dei nodi.

5. **Ricerca delle chiavi** : ogni nodo, scelto secondo l'ordine di immissione nella rete, esegue un'operazione di GET verso tutti i nodi della rete. Il totale delle operazioni di GET effettuate dall'intera DHT è uguale al quadrato del numero di nodi presenti.

6. **Mostrare lo stato**: ogni nodo emette in output le informazioni relative alle sue strutture dati, la routing table e la hit table

Ho eseguito più simulazioni basate su questo esperimento variando il numero di nodi inseriti nella rete: sono state realizzate prove con 256, 512 e 1024 nodi.

Ho eseguito dieci esecuzioni per ogni esperimento ed in tutti i casi i nodi presentavano sia il medesimo numero di query totali inoltrate che la medesima distribuzione. I valori ottenuti sono mostrati in Tabella 1 e nel Grafico 17.

Da notare che la distribuzione dei dati rimane identica al variare del numero dei nodi: un picco iniziale, una parte crescente ed infine una parte decrescente. A parte il massimo locale centrale, i valori corrispondenti delle varie prove sono perfettamente correlate: il valore raddoppia al raddoppiare dei nodi. Il valore che si “aggiunge” all'aumentare del numero di nodi è quello a cui corrisponde il massimo locale.

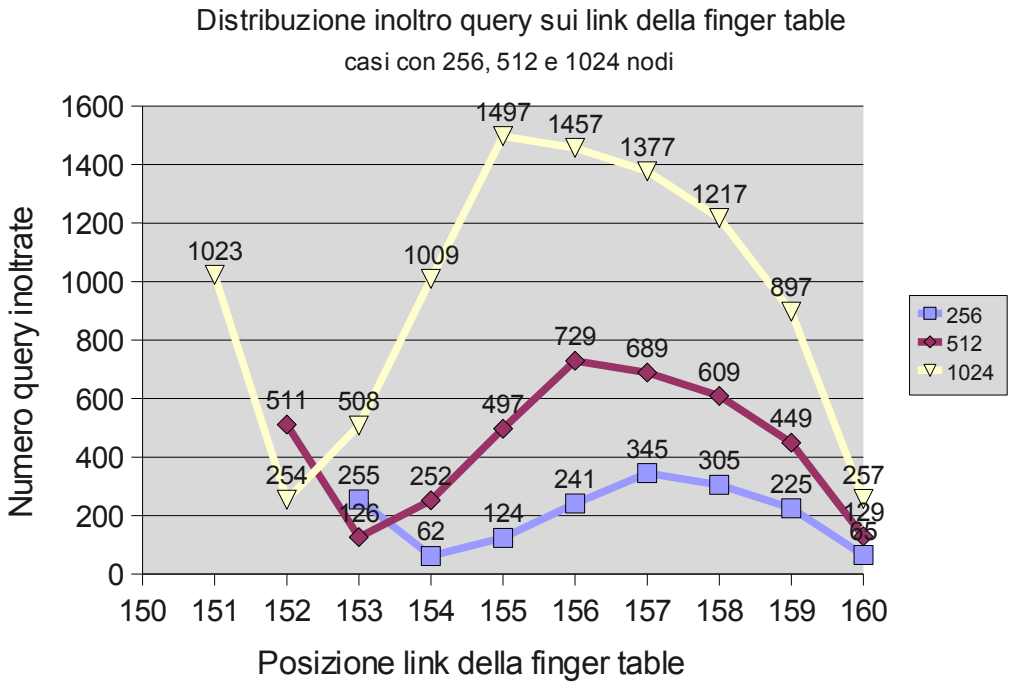


*Grafico 17: Distribuzione degli inoltri sui link della finger table di un nodo chord al variare del numero di nodi nella rete*

| Distribuzione query sui link della tabella di routing | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Numero Nodi** | **Link tabella di routing** | | | | | | | | | |
| | **150** | **151** | **152** | **153** | **154** | **155** | **156** | **157** | **158** | **159** |
| **256** | | | 255 | 62 | 124 | 241 | 345 | 305 | 225 | 65 |
| **512** | | 511 | 126 | 252 | 497 | 729 | 689 | 609 | 449 | 129 |
| **1024** | 1023 | 254 | 508 | 1009 | 1497 | 1457 | 1377 | 1217 | 897 | 257 |

*Tabella 1: Distribuzione delle query sui link della finger table. Notare come, ad esclusione del valore centrale, i valori corrispondenti delle diverse righe raddoppino al raddoppiare dei nodi*

## 4.6 - Secondo esperimento con routing stabilizzato

Questo esperimento analizza una rete in cui tutti i nodi sostengono il medesimo carico, ma alcuni di essi sono concentrati in una sezione dello spazio degli indirizzi creando così un hot spot. Lo scopo dell'esperimento è di analizzare come la presenza di una tale zona influisca sulla distribuzione dell'inoltro delle query sui link degli altri nodi della rete. Uno dei passi dell'esperimento assicura la stabilizzazione delle finger table di tutti i nodi.

Per questo esperimento ho utilizzato Overlay Weaver configurato in modo da utilizzare l'algoritmo di routing Chord e il sistema di inoltro dei messaggi di tipo ricorsivo.

Lo scenario che descrive la simulazione prevede i seguenti passi:

1. **Inserimento dei nodi**: Sono generati $n+k$ nodi, che ricevono un ID scelto casualmente fra due sottoinsiemi. Il primo sottoinsieme contiene $n$ ID che sono disposti equamente nello spazio degli identificatori, il secondo sottoinsieme contiene degli ID disposti in modo da popolare la sezione dello spazio degli identificatori fra due nodi successivi. Il numero $k$ di nodi inseriti nella sezione più popolata, assume uno dei seguenti valori: 1, 2, 4, 8, 16, 32, 64. I nodi per il loro inserimento utilizzano un peer scelto casualmente fra quelli già inseriti nella rete. Il lasso di tempo che intercorre fra due inserimenti è di 200 millisecondi.

2. **Introduzione delle chiavi**: su ogni nodo è effettuata un'operazione di PUT di una chiave il cui ID è uguale all'identificatore del nodo. In questo modo la PUT non esegue routing per completare l'inserimento della chiave.

3. **Stabilizzazione della finger table**: questo passaggio assicura che ogni nodo abbia in tutti i link della finger table un collegamento verso un nodo della rete. Per fare ciò su ogni nodo si esegue una GET verso tutti gli altri nodi. Questo passaggio è stato sovradimensionato rispetto alla necessità per avere la certezza di una rete Chord stabile.

4. **Azzeramento valori** : Il passo precedente utilizza l'operazione di GET per stabilizzare la tabella di routing. Per evitare che questi valori influiscano sulle analisi, in questo passo si cancellano tutte le hit table dei nodi.

5. **Ricerca delle chiavi**: ogni nodo, scelto secondo l'ordine di immissione nella rete, esegue un'operazione di GET verso tutti i nodi della rete. Il totale delle operazioni di GET effettuate dall'intera DHT è uguale al quadrato del numero di nodi presenti.

6. **Mostrare lo stato**: : ogni nodo emette in output le informazioni relative alle sue strutture dati, la routing table e la hit table

I dati sono stati acquisiti con le stesse modalità dell'esperimento precedente: nella posizione $ht_i$ della hit table è registrato il valore corrispondente al numero di messaggi di GET inoltrati attraverso il link della finger table $rt_i$ .

I dati ottenuti dalle simulazioni con 256 nodi sono mostrati nei grafici dal 18 al 24 , e mettono in evidenza l'incidenza di un hot spot sulla distribuzione delle query fra i link della finger table dei nodi. Ogni linea di un grafico mostra la distribuzione delle query dei nodi che raggiungono lo hot spot tramite il link della finger table che da il nome alla curva. Si nota come nei casi con hot spot di intensità ridotta con un sovraccarico di al massimo due volte la media, grafici 18 e 19, gli effetti sulla distribuzione siano limitati, mentre nei grafici successivi (20, 21, 22, 23 e 24) che mostrano hot spot con un sovraccarico di almeno quattro volte la media, l'influenza sulla distribuzione è molto consistente: per esempio nel grafico 20 in cui lo hot spot è 4 volte il valore medio, la linea contrassegnata con 156 nel suo punto di hot spot presenta un incremento di quasi il 50%. Voglio ricordare che dato il numero di nodi, il consistent hashing in condizioni normali, presenta uno sbilanciamento proporzionale a $\log(256)=8$ volte la media.

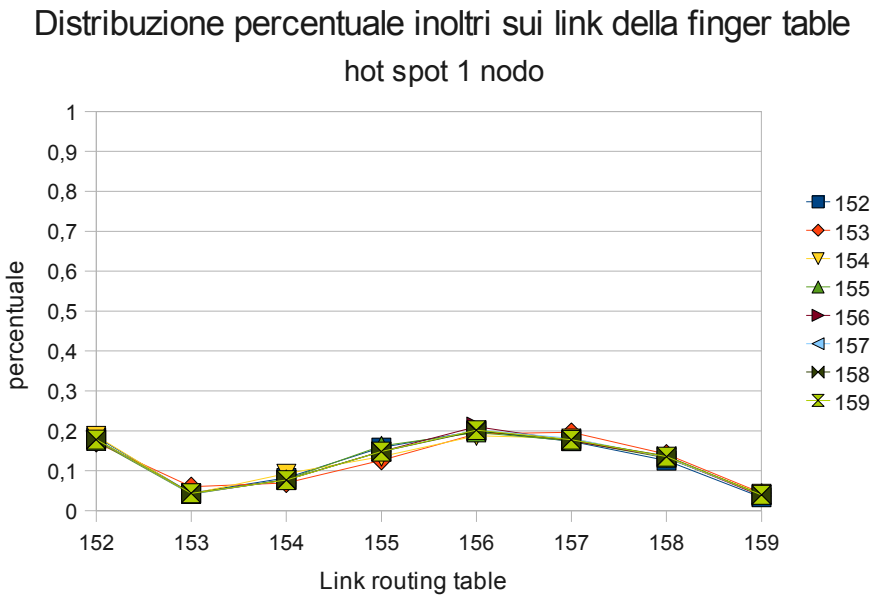


*Grafico 18: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

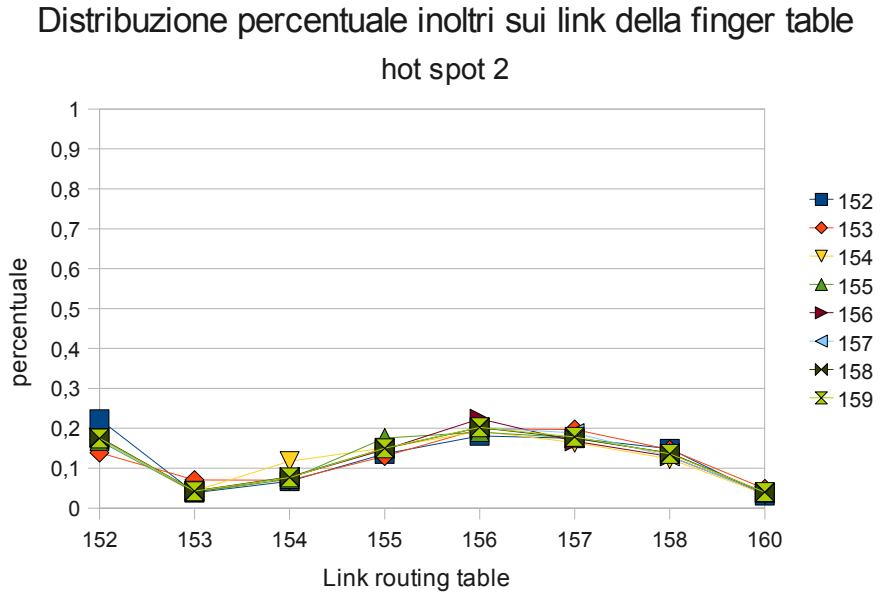


*Grafico 19: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

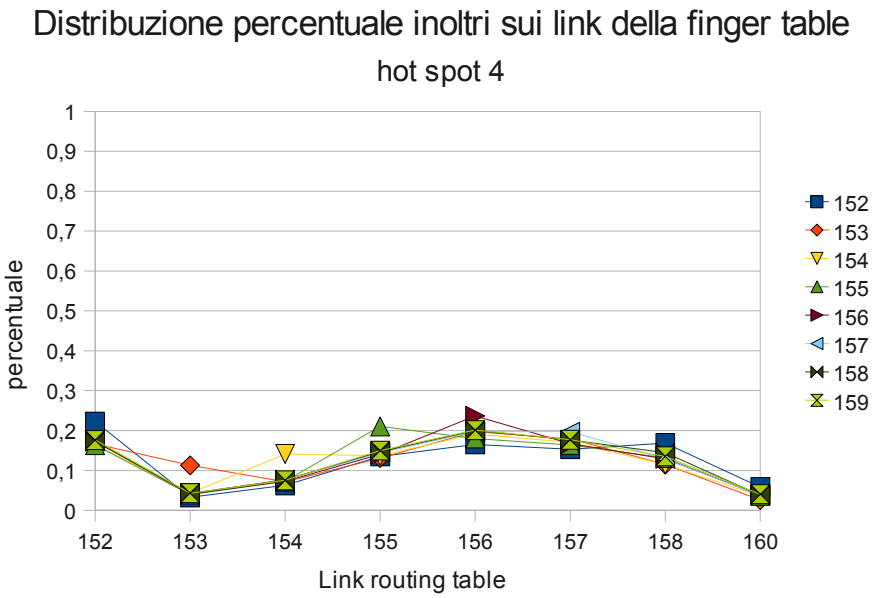


*Grafico 20: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

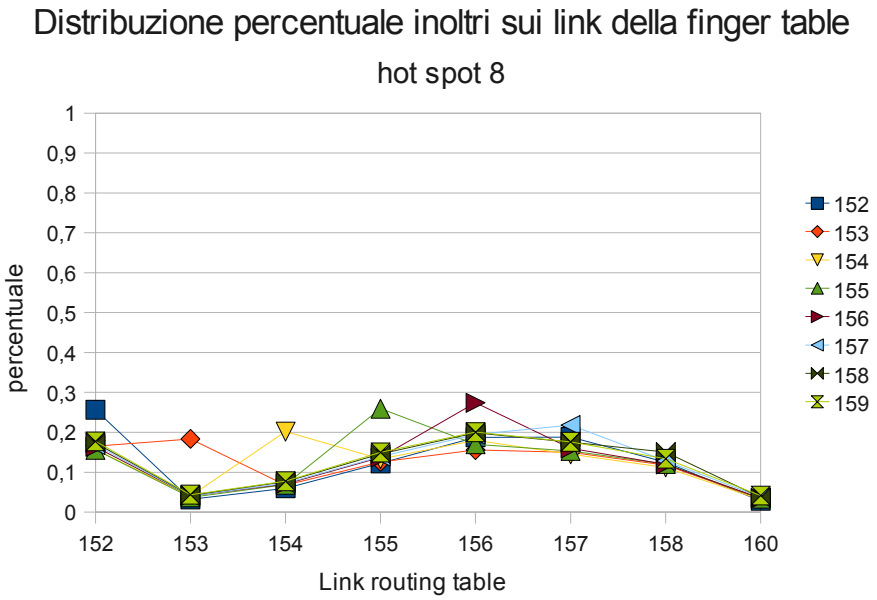


*Grafico 21: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

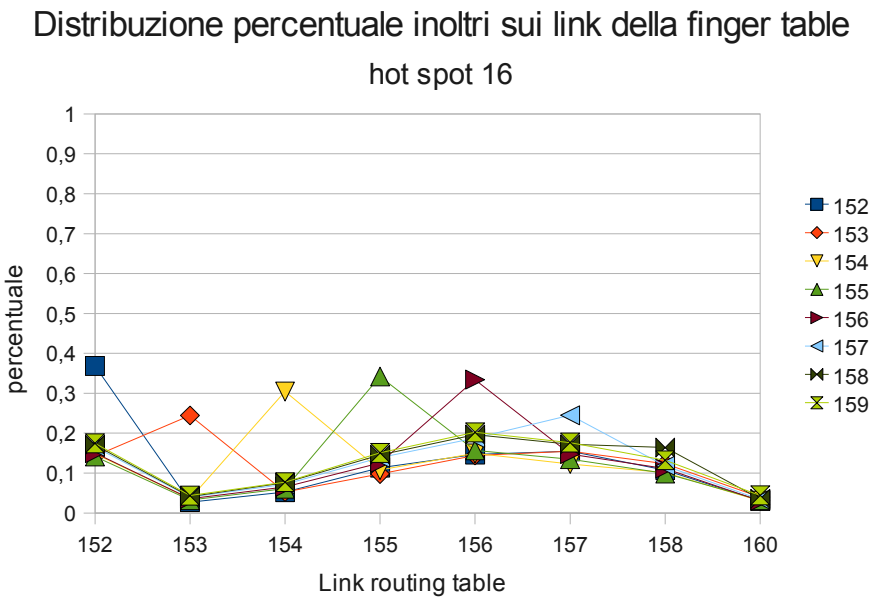


*Grafico 22: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

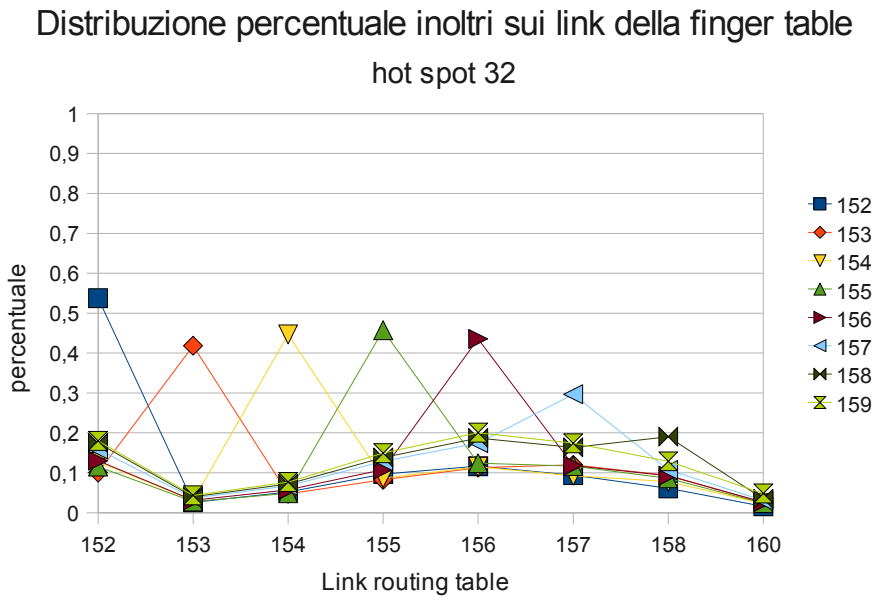


*Grafico 23: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

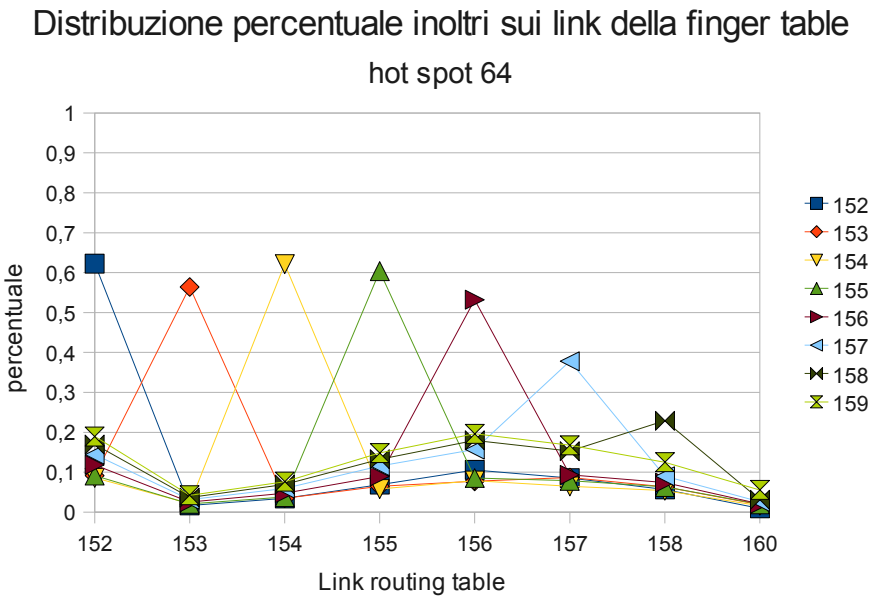


*Grafico 24: distribuzione percentuale dei messaggi inoltrati attraverso i link della finger table di un nodo*

## 4.7 - Terzo esperimento con routing stabilizzato

L'esperimento precedente crea un hot spot incrementando il numero di nodi, ognuno con il medesimo carico di lavoro, in una sezione dello spazio degli identificatori. In questo esperimento lo hot spot è creato aumentando il carico di un nodo: uno dei nodi gestisce un maggior numero di chiavi ognuna con associato il medesimo carico di tutte le altre chiavi della rete.

Lo scopo di questo esperimento è analizzare se sono presenti delle variazioni sulla distribuzione degli inoltri delle query rispetto al caso precedentemente studiato.

Per questa simulazione ho creato uno scenario che esegue i seguenti passi:

1. **Inserimento dei nodi**: Sono creati $n$ nodi che eseguono l'operazione di join. L'ID dei nodi è assegnato tramite la formula $\frac{2^{160}}{n} * i$ con $i = 0, 1, ..., n-1$ , ma l'ordine di inserimento della rete è casuale in modo da non condizionare la creazione di link stabilendo un ordine preciso. Ogni nodo si inserisce nella rete tramite un altro peer scelto casualmente fra quelli già presenti. Il lasso di tempo che intercorre fra due inserimenti è di 200 millisecondi.

2. **Introduzione delle chiavi**: su ogni nodo è effettuata un'operazione di PUT di un chiave il cui ID è uguale all'identificatore del nodo. In questo modo la PUT non esegue routing per completare l'inserimento della chiave. Il nodo con ID 0 esegue un numero $k$ di ulteriori operazioni di PUT che gli assegneranno la gestione di un maggior carico di lavoro.

3. **Stabilizzazione della finger table**: questo passaggio assicura che ogni nodo abbia in tutti i link della finger table un collegamento verso un nodo della rete. Per fare ciò su ogni nodo si esegue una GET verso tutti gli altri nodi. Questo passaggio è stato sovradimensionato rispetto alla necessità per avere la certezza di una rete Chord stabile.

4. **Azzeramento valori** : Il passo precedente utilizza l'operazione di GET per stabilizzare la tabella di routing. Per evitare che questi valori influiscano sulle analisi, in questo passo si cancellano tutte le hit table dei nodi.

5. **Ricerca delle chiavi**: ogni nodo, scelto secondo l'ordine di immissione nella rete, esegue un'operazione di GET verso tutti i nodi della rete. Il totale delle operazioni di GET effettuate dall'intera DHT è uguale al quadrato del numero di nodi presenti.

L'esperimento è stato eseguito con diversi valori di $k$ : 1, 2, 4, 8, 16, 32, 64.

L'analisi della distribuzione delle query sui link ha dato come risultato una curva del tutto analoga a quella ottenuta dal secondo esperimento, sia per forma che per valori. Questo esperimento è stato comunque molto utile perché ha messo in evidenza una caratteristica del link che collega il peer al suo successore, ovvero che il numero di messaggi inoltrati attraverso di esso è fortemente correlato con il numero di query gestite dal successore del nodo: il valore è quasi identico al numero di query ricevute.

Per dare maggiore peso a questa osservazione qualitativa, si è deciso di calcolare il coefficiente di correlazione fra il numero di query che sono osservate sul link che collega il nodo $i-1$ al nodo $i$ , ed il numero di query ricevute dal nodo $i$ in tutti gli esperimenti effettuati. Il risultato è stato che il valore dell'indice di correlazione è:

$$\rho_{xy} = 0{,}99$$

che denota una corrispondenza quasi perfetta fra l'andamento delle due variabili.

## 4.8 - Fitting dei dati

Al fine di poter fornire una buona stima di $\overline{ht_i}$ è necessario determinare una funzione che ne permetta di calcolare il valore. Poiché la distribuzione del carico non sembra seguire nessuna funzione "comune" come un esponenziale o una parabola, è stato utilizzato uno strumento di analisi di regressione che potesse eseguire il fit dei punti, ottenuti dagli esperimenti con routing stabilizzato, analizzando un elevato numero di funzioni.

Lo strumento utilizzato è stato Datafit [27], programma che esegue una analisi di regressione [28] dei dati in input sia lineare che non lineare utilizzando l'algoritmo Levenberg–Marquardt [29] [30]. Lo scopo di un'analisi dei dati di questo tipo è determinazione della curva che permette di calcolare i punti con la migliore approssimazione possibile. Il programma restituisce oltre la funzione che descrive la curva, anche il valore dei parametri che permettono alla curva di acquisire i valori ricercati.

Ovviamente utilizzando un polinomio di grado sufficiente è sempre possibile avere una curva che passa attraverso tutti i punti, ma ciò richiederebbe avere un numero molto elevato di parametri rendendo inutile un'analisi di questo tipo.

Per cui lo scopo dell'analisi di regressione per eseguire il curve fitting è quello di stabilire la curva che meglio si adatta ai dati forniti con un numero ragionevole di parametri.

Il criterio per quantificare la bontà del fitting dei dati tramite una determinata funzione è di fondamentale importanza per poter scegliere la migliore: una data funzione potrebbe fornire degli ottimi risultati per alcuni punti mentre per altri fornire una approssimazione grossolana, mentre un'altra funzione potrebbe approssimare tutti i valori discostandosi poco da essi. Come fare a stabilire quale delle due curve è migliore?

Nell'algoritmo Levenberg–Marquardt utilizzato in Datafit, si utilizza il criterio dei minimi quadrati [31]: il fitting della curva sui dati è considerato tanto migliore

quanto è minore la somma dei quadrati degli scarti fra il dato ed il valore calcolato dalla curva ovvero la curva migliore è quella che minimizza:

$$s(\beta)=\sum_{i=1}^{m}(y_i-f(x_i,\beta))^2$$

dove $x_i, y_i$ sono rispettivamente la variabile indipendente e la variabile dipendente, ovvero i dati di cui eseguire il fitting, $f()$ è la funzione con la quale si sta cercando di approssimare la curva e $\beta$ sono i suoi parametri.

I parametri $\beta$ della curva sono un parametro dell'operazione di fitting: l'algoritmo Levenberg–Marquardt calcola il loro valore tramite un processo di approssimazione iterativo, che in alcuni casi può divergere facendo fallire l'operazione di fitting.

Datafit ha provato circa trecento possibili funzioni non lineari sull'insieme di dati, ma i risultati ottenuti dal fitting sono stati deludenti: la migliore curva ottenuta presenta un valore calcolato di $s(\beta)$ molto alto che indica un fitting piuttosto grossolano con scostamenti sensibili fra i punti calcolati e quelli rilevati.

Per questo motivo i punti sono stati analizzati spezzando i dati in tre porzioni: la sezione crescente, la parte decrescente, ed il primo valore della curva. Quest'ultimo rappresenta il collegamento con il prossimo nodo ed ho deciso di non considerarlo nell'operazione di fit in quanto dovrebbe può essere comunque trattato in modo separato, come rilevato dalle osservazioni fatte a seguito del terzo esperimento.

Analizzando i valori della parte decrescente della curva è stato possibile osservare come questa sia in realtà un'esponenziale negativa infatti si ha che $\log(x_{i+2}-x_{i+1})-\log(x_{i+1}-x_i)$ è un valore costante, caratteristica tipica di una funzione esponenziale.

La parte crescente della curva è stata approssimata utilizzando la retta ottenuta applicando la regressione lineare ai dati, ma in tutti e tre i casi (256, 512 e 1024 nodi) i punti centrali risultavano sovrastimati, mentre i punti esterni risultavano sottostimati. Nonostante gli scarti fossero di ampiezza ridotta ho preferito cercare una curva che si adattasse meglio alla distribuzione dei punti: la metodicità del

segno degli scarti porta a pensare che la curva approssimante non stia descrivendo la reale "legge" sottostante. Infatti un indicatore qualitativo di una funzione di fitting è che il segno degli scarti sia distribuito in modo casuale.

Per questo motivo ho cercato una curva non lineare, che pur approssimando in modo molto migliore i punti, richiede un parametro in più.

La funzione utilizzata per eseguire il fitting della curva è stata :

$$\overline{ht_i} = \begin{cases} b * e^{(c*(i-i_{max}))} + d - b & i \geq i_{max} \\ \dfrac{b_1}{1 + e^{(b_2 - b_3 * (i - i_{max}))}} & i < i_{max} \end{cases}$$

Dove $i_{max}$ è l'indice del valore $ht_i$ massimo, utilizzato come punto di divisione fra la parte crescente della curva [32] e la parte decrescente .

Utilizzando questa curva Datafit non è stato in grado di portare a termine il fitting: con il valore di default assegnato ai parametri il processo iterativo divergeva.

Per cui è stato necessario attribuire un valore iniziale ai parametri attraverso uno studio della curva per assegnare ai suoi parametri dei valori che seguono le caratteristiche della distribuzione dei dati analizzata.

Il valore di $c$ è stato approssimato con $\log(x_{i+2} - x_{i+1}) - \log(x_{i+1} - x_i)$ in quanto quest'ultimo valore è legato alla base dell'esponenziale, mentre Il valore di $d$ è stato approssimato con il valore massimo della curva in quanto è il valore assunto da $F1$ in quel punto.

Assegnando questi valori, l'operazione di curve fitting è riuscita a convergere definendo così il valore di tutti i parametri:

| Parametro | b | b1 | b2 | b3 | c |
|---|---|---|---|---|---|
| Valore | -0,0032 | 0,8468 | 1,1818 | 0,7340 | 0,7346 |

*Tabella 2: Tabella dei parametri della curva di fit*

La bontà del fitting è molto soddisfacente: i valore in termini assoluti di $s(\beta)=1{,}8902*10^{-05}$ è decisamente molto basso. Il Grafico 25 mostra l'approssimazione dei punti rilevati attraverso l'utilizzo della funzione calcolata. Si può notare come la distanza fra i punti e la curva sia estremamente ridotta.

Non è stato necessario eseguire un'analisi sulla varianza degli scarti, come è stato fatto nella prima serie di test, in quanto a differenza di quei test i dati presentano una varianza uguale a 0: ogni simulazione del primo esperimento della seconda serie di test ha presentato dei risultati fra di loro identici, per cui i dati utilizzati durante l'operazione di fitting non sono il risultato di una media, ma esattamente i valori rilevati.

La funzione ottenuta è in grado di approssimare il valore di $\overline{ht_i}$ commettendo un errore molto ridotto, quindi utilizzabile durante l'analisi del livello di carico di un link della finger table di un nodo.

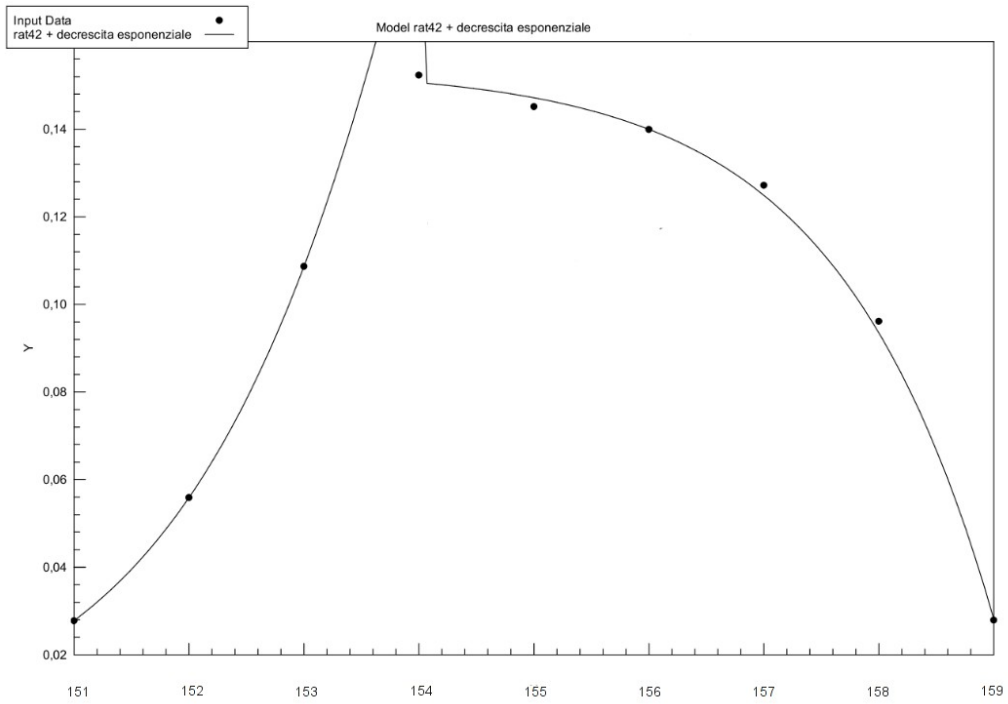


*Grafico 25: Curva di approssimazione della distribuzione dei link nella routing table*

# Capitolo 5 - Strumenti utilizzati

L'analisi della distribuzione delle query sui link della routing table di un nodo, è stata affrontata utilizzando una rilevazione dei dati ottenuti dall'esecuzione di simulazioni. In questo capitolo descrivo i due programmi utilizzati per costruire una rete DHT sulla quale eseguire i test, ed i problemi che hanno presentato, come il già citato bug di aggiornamento delle finger table di Overlay Weaver.

Il primo programma utilizzato è Bamboo DHT [24] [33], programma alla base di OpenDHT un servizio che mantiene una DHT utilizzata per scopi accademici con hardware fornito da PlanetLab. Questo programma implementa una rete di tipo Pastry.

Pur essendo uno strumento molto utilizzato che fornisce sia una buona utilizzabilità sia delle buone prestazioni, si è dimostrato poco incline alla modifica: la sua implementazione è molto confusionaria, spesso incoerente con se stessa, problema derivante a mio parere da modifiche al codice eseguite senza rispettare l'ingegnerizzazione di base.

Nonostante i grandi sforzi fatti per capire la struttura del codice sorgente e per scrivere le modifiche necessarie all'implementazione degli algoritmi di rilevazione degli hot spot, ho riscontrato problemi tali da convincermi a sostituirlo.

Il secondo programma utilizzato è stato Overlay Weaver [25] [34] che basa la sua struttura sulla separazione logica del codice necessario all'implementazione degli overlay dal codice necessario per implementare le altre caratteristiche del programma: ciò fornisce la possibilità di implementare un nuovo tipo di rete scrivendo solo poche centinaia di righe di codice. Insieme al programma sono distribuite le implementazioni di cinque reti DHT molto note: Chord, Pastry, Kademlia, Koorde e Tapestry.

Nel lavoro svolto per questa tesi, oltre ad estendere questi due strumenti per poter includere il supporto agli algoritmi di individuazione degli hot spot,è stato

necessario implementare una serie di utility per la creazione delle simulazioni e per l'analisi dei dati.

In particolare ho implementato uno strumento che permette di simulare un routing basato sul carico a partire dai dati ricavati dall'esecuzione di una simulazione eseguita con Overlay Weaver: questa applicazione è stata fondamentale per testare e validare l'algoritmo di individuazione degli hot spot.

## 5.1 - Bamboo DHT

Bamboo DHT [34] è un programma che implementa una DHT con geometria Pastry ponendo particolare attenzione alla gestione di un elevato churn rate [24].

Un'istanza di Bamboo, che partecipa ad una rete P2P, fornisce l'accesso ai dati della DHT ad altre applicazioni anche se eseguite su dei PC remoti. Ciò è realizzato grazie all'implementazione di un Gateway Sun RPC che permette di eseguire le comuni operazioni di una DHT come l'inserimento, la ricerca e la rimozione di una chiave. Il protocollo utilizzato dal gateway è implementato sopra il protocollo TCP/IP che lo rende facilmente accessibile da macchine remote.

Bamboo è un programma modulare composto da diversi componenti chiamati stage, ognuno dei quali si occupa della gestione di una funzionalità del peer. Un'istanza di Bamboo può selezionare, attraverso un apposito file di configurazione, quali stage caricare indicando anche dei dati di inizializzazione. Per esempio, in questo file è indicato l'indirizzo dei nodi che fungono da punto di accesso alla rete DHT della quale si desidera che il nodo entri a far parte.

La creazione di un nuovo stage è un'operazione che richiede l'implementazione di poche interfacce e la definizione dei dati che saranno gestiti dal componente. La creazione o la modifica di uno stage non richiede che il resto dell'applicazione venga ricompilato: Bamboo caricherà dinamicamente tale stage durante l'avvio del programma se presente nel file di configurazione. Il sistema di componenti di Bamboo permette di selezionare l'insieme degli stage utilizzati rendendo possibile la personalizzazione del comportamento di tale strumento.

Gli stage sono in grado di comunicare fra di loro attraverso lo scambio di messaggi. Essi, durante la propria esecuzione, possono registrarsi per la ricezione dei messaggi originati da altri stage. La gestione di tali messaggi è eseguita attraverso una coda associata allo stage nella quale è inserita una copia di tutti i messaggi a cui si è eseguito la sottoscrizione.

Gli stage, durante la propria inizializzazione, definiscono anche un elenco che contiene la tipologia dei messaggi generati ai quali altri stage possono sottoscriversi.

Gli stage necessari per svolgere il compito richiesto ad un peer completo sono:

1. **network**: gestisce i messaggi di rete tcp
2. **router**: permette lo scambio dei messaggi fra i nodi, utilizzando la geometria della rete
3. **datamanager**: registra le chiavi che il nodo deve salvare
4. **storagemanager**: permette la duplicazione delle chiavi fra i nodi del leafset
5. **DHT**: effettua un'astrazione delle funzionalità della DHT, permettendo le operazioni di PUT e di GET. Si basa sullo stage di routing.
6. **gateway**: implementa il gateway RPC che permette l'invocazione da remoto di operazione di PUT e di GET.

Bamboo fornisce anche altri stage non necessari per l'esecuzione di un peer pastry che forniscono caratteristiche opzionali come per esempio la visualizzazione dello stato di un nodo attraverso accesso web.

L'architettura a stage è implementata in Bamboo attraverso SEDA (Staged, Event Driven Architecture) [35] un sistema che permette di ridurre la complessità di un'applicazione attraverso la decomposizione in componenti. Questo design basato sullo scambio di messaggi, è stato sviluppato per sistemi multiprocessore: esso permette di raggruppare i componenti in modo che siano eseguiti in un numero di thread uguale al numero di unità di esecuzione di cui è fornito l' hardware.

In tal modo è possibile utilizzare le caratteristiche di un sistema multi core o multiprocessore minimizzando l'overhead dovuto alla rischedulazione per l'esecuzione dei thread.

L'autore di Bamboo, partendo dall'osservazione che i computer normalmente diffusi durante la scrittura del suo programma erano dotati di un unico processore

single core, ha deciso di implementare Bamboo in modo da poter essere eseguito in un unico thread [36]. Sono comunque previsti l'utilizzo di thread separati per gestire componenti esterni caratterizzati da un design che esegua le operazioni “lente” in modo bloccante, come ad esempio il Berkley data base utilizzato nello stage data manager.

Il codice di Bamboo è stato sottoposto ad un porting non completato e non più in corso, del sistema di messaggi. I suoi componenti di base sono stati convertiti al più efficiente SFS [37] che si basa sul meccanismo di callback, mentre l'architettura ed alcuni componenti sono rimasti legati al sistema SEDA. Ciò ha influito molto negativamente sulla leggibilità del codice, in quanto operazioni simili sono spesso implementate in modo differente, causando anche replicazioni di codice.

### 5.1.1 - Rilevazione dei dati da una simulazione

Ho utilizzato Bamboo per creare delle simulazioni per l'acquisizione dei dati allo scopo di individuare la legge che descrive la distribuzione degli inoltri delle query fra i link della routing table.

L'operazione di creazione degli esperimenti non è stata del tutto banale: Bamboo non prevede né una modalità “simulazione” , né un meccanismo che permetta di descrivere facilmente le operazioni da far eseguire ai nodi della rete. Per cui è stato necessario costruire una rete reale di nodi di Bamboo in cui le operazioni da eseguire sono state impartite attraverso l'utilizzo del gateway eseguito dal peer.

Per sopperire all'assenza di un sistema che permetta di descrivere facilmente sia quanti nodi caricare in una rete e come distribuirli nei nodi fisici, sia le operazione che devono essere compiute, ho implementato due utility.

La prima di queste utility, implementata con uno script Bash, esegue il numero indicato di nodi Bamboo distribuiti equamente su un elenco di macchine fornito in un apposito file di configurazione.

La seconda utility utilizza un ulteriore file di configurazione in cui sono descritte le operazioni che devono essere eseguite sull'elenco di peer. I nodi della rete ricevono le operazioni di inserimento e di ricerca delle chiavi attraverso il loro gateway.

Le simulazioni effettuate con Bamboo sono state eseguite su reti con un basso numero di nodi: esse contengono soltanto 256 nodi, in quanto le risorse hardware dei PC a utilizzati non sono in grado di gestire più di 16 istanze di Bamboo.

La simulazione eseguita prevede una distribuzione del tutto simile a quella descritta dal primo esperimento del capitolo 4: i nodi ricevono un indirizzo al momento dell'inserimento della rete che li posiziona in modo che le distanze fra ogni coppia di nodi successivi sia costante.

Con una tale distribuzione dei nodi nello spazio degli identificatori, le tabelle di routing dei peer presentano delle tabelle di routing in cui soltanto le prime due righe contengono dei link ad altri nodi: una quantità piuttosto ridotta per poter cercare di estrapolare la legge che ne descrive il comportamento del sistema di routing.

L'incremento del numero dei nodi della simulazione non è stato possibile in quanto per riempire un'ulteriore riga della tabella di routing sarebbero stati necessarie 4096 istanze di Bamboo, che richiedevano un numero troppo elevato di PC.

Data l'impossibilità di agire in questo modo si è pensato di intervenire su un parametro tipico della geometria Pastry: il parametro $b$, ovvero il parametro che determina la base delle “cifre” di un identificatore Pastry.

In Bamboo il valore di default della base delle cifre è $2^b = 16$, diminuendo il valore di $b$ da 4 a 2, la dimensione di una riga della tabella diminuisce sensibilmente aumentando il numero di righe “complete” della tabella di routing.

Per poter analizzare la distribuzione dei messaggi inoltrati, ho dapprima intercettato il messaggio che richiede allo stage di routing di inoltrare un messaggio al prossimo nodo. Questa tecnica però non ha funzionato in quanto

Bamboo replica tutto il sistema di routing nello stage DHT per i soli messaggi di PUT, GET e REMOVE delle chiavi.

Eseguire la modifica per permettere al sistema di rilevazione degli inoltri delle query di analizzare il traffico sullo stage DHT è stato molto laborioso: questo stage è il risultato di molti interventi al codice che non hanno seguito le più basilari tecniche di buona programmazione.

Per esempio molte variabili il cui valore è assegnato attraverso il file di configurazione del peer durante l'inizializzazione dello stage ignorano tale valore solo per alcuni utilizzi di tale variabile, utilizzando al suo posto dei valori assegnati manualmente in porzioni del codice diverse da quelle che eseguono l'inizializzazione del nodo.

Un altro esempio è dovuto al valore di alcune costanti contenenti dei magic number utilizzate in contesti differenti e con significati diversi da quelli definiti dal nome della costante. È inoltre presente una elevata quantità di codice replicato, di cui le varie copie si differenziano soltanto per il valore di una variabile.

Questi sono soltanto alcuni problemi riscontrati che, unite ad un codice scarsamente leggibile e poco documentato, rendono estremamente complesso eseguire le modifiche richieste per la rilevazione della distribuzione delle query.

Analizzando il codice dello stage DHT è stato possibile esaminare anche il reale algoritmo utilizzato per l'inoltro delle operazioni di PUT, GET e REMOVE.

Infatti a differenza di tutti gli altri messaggi il cui algoritmo di routing è definito nello stage di routing e segue il protocollo definito nell'articolo di Pastry, questi messaggi sono gestiti in modo differente: il nodo successivo è selezionato, oltre che per la distanza del routing anche per il tempo di risposta di quest'ultimo [38].

Questa variazione dell'algoritmo di routing è introdotta come sistema di bilanciamento del carico: un nodo lento nella risposta è probabilmente un nodo sovraccarico, per cui anziché inoltrargli una query, la si spedisce seguendo un altro percorso più scarico, ma più lungo.

Questo intervento modifica direttamente il comportamento del routing modificando la distribuzione degli inoltri attraverso i link della routing table, per cui è incompatibile con il sistema di rilevazione del carico studiato in questo lavoro di tesi.

Dato lo stato del codice di Bamboo, piuttosto che modificare lo stage DHT per utilizzare il normale algoritmo Pastry già implementato nello stage di routing, ho preferito abbandonare questo strumento in favore di Overlay Weaver.

## 5.2 - Overlay Weaver

Overlay Weaver [25] [34] è uno strumento per la creazione ed il confronto di overlay. Permette di implementare una nuova rete scrivendo soltanto poche centinaia di righe di codice Java: la realizzazione di un nuovo overlay avviene tramite l'implementazione di alcune interfacce che si limitano a definire le parti essenziali come la geometria utilizzata e l'algoritmo di routing. Le altre componenti tipiche di un sistema P2P come la gestione dei socket, la spedizione fisica dei messaggi o la gestione delle chiavi, sono implementati direttamente dallo strumento e la loro complessità è totalmente mascherata dalle interfacce fornite per il loro utilizzo. Overlay Weaver permette comunque di personalizzare anche le operazioni non necessariamente collegate alle operazioni di routing: è sufficiente implementare le suddette interfacce relative alle funzionalità che si desidera modificare.

Nell'Illustrazione 7 si mostra l'architettura di Overlay Weaver. Nel riquadro che indica il routing layer sono mostrati tutti i componenti che dipendono dall'overlay prescelto. Tuttavia per implementare un nuovo overlay è sufficiente fornire un'implementazione della sola interfaccia routing Algorithm.

Alcuni overlay richiedono che gli inoltri dei messaggi seguano un protocollo stabilito: per esempio Chord richiede che gli inoltri siano di tipo ricorsivo, ovvero il nodo che riceve un messaggio da inoltrare si occupa di spedire il messaggio al nodo successivo.

L'altra possibilità è il routing di tipo iterativo nel quale un nodo che riceve la richiesta di inoltro di un messaggio spedisce una risposta al nodo che ha originato la richiesta che si occuperà di far proseguire il routing.

Overlay Weaver non si limita ad offrire un'implementazione per entrambe le strategie, ma permette all'utilizzatore di fornire una propria strategia mediante l'implementazione dell'interfaccia Routing Runtime.

Un'altra sezione interessante è l'interfaccia del Messanger service, che si occupa di astrarre il mezzo di trasmissione per le comunicazioni: Overlay Weaver

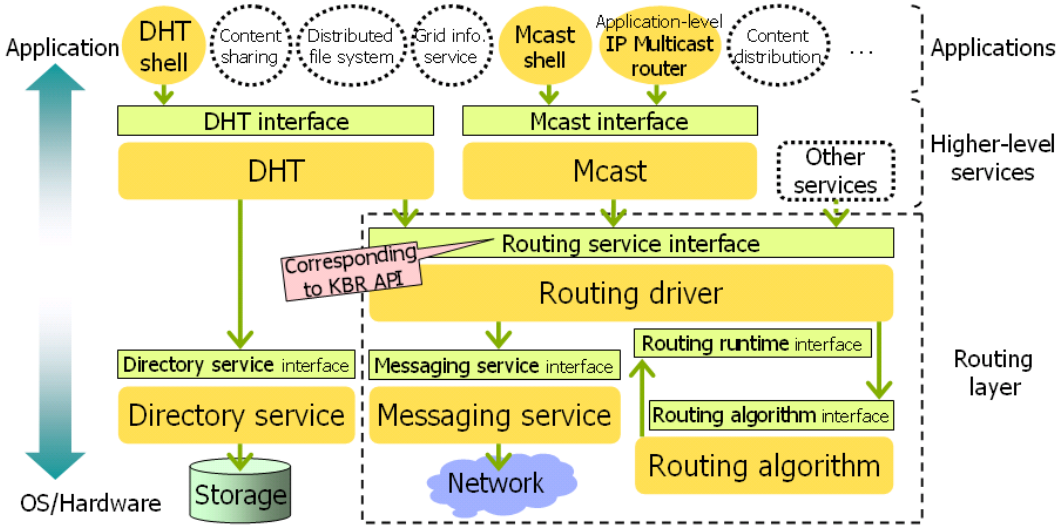


*Illustrazione 7: Architettura di Overlay Wearver: per implementare un nuovo overlay è sufficiente implementare la Routing Algorithm interface.*

offre l'implementazione dei backend per il protocolli TCP ed UDP, e fornisce anche il supporto ad un veloce sistema di comunicazione per le simulazioni.

Quest'ultimo backend permette di fornire il supporto alla simulazione di reti DHT eseguite su un unico computer in modo molto efficiente: Overlay Weaver è infatti in grado di simulare reti con decine di migliaia di nodi su un normale PC. Ciò lo rende un ottimo strumento per lo sviluppo di un nuovo overlay, infatti dopo la fase di ricerca potrà essere utilizzato immediatamente per costruire una rete reale: Overlay Weaver non richiede nemmeno di dover eseguire una ricompilazione del codice sorgente per eseguire in una rete reale l'overlay prodotto e testato con delle simulazioni.

Queste caratteristiche lo rendono un ottimo strumento per confrontare overlay differenti: le uniche parti che si differenziano fra gli overlay da analizzare sono gli algoritmi che caratterizzano il sistema di routing. Un tale utilizzo è reso ancora più efficace dalla possibilità di definire l'elenco delle operazioni da eseguire e l'istante in cui devono essere eseguiti nella simulazione. Per utilizzare questa funzionalità è sufficiente scrivere un file, detto scenario, che elenca le operazioni richieste.

L'insieme delle operazioni che possono essere introdotte nel file di scenario include tutte le normali operazioni di una rete DHT come le operazioni di JOIN, PUT e GET, ed anche dei comandi per analizzare lo stato dei nodi. È fornita anche la possibilità di estendere l'insieme delle operazioni implementando un'apposita interfaccia.

### 5.2.1 - Problemi riscontrati

Utilizzando Overlay Weaver ho riscontrato alcuni problemi che hanno influito sulla possibilità di eseguire i test descritti nel capitolo precedente.

Il primo problema è stato l'impossibilità di eseguire simulazioni con più di 1024 nodi. Seppure Overlay Weaver sia in grado di eseguire delle simulazioni con un numero molto maggiore di nodi simulati, il design della gestione dei file di scenario ha un limite notevole: le simulazioni prima della loro esecuzione devono caricare integralmente il file di scenario che le descrive in memoria.

La dimensione di un file di scenario che descrive una simulazione con 2048 nodi con caratteristiche simili a quelle mostrate negli esperimenti del capitolo precedente, è superiore al giga byte. La macchina di test non disponeva di sufficiente memoria per completare la simulazione.

Una dimensione così elevata del file è causata dalla combinazione di due fattori:

1. la lunghezza della chiave: la chiave, nell'operazione di GET, deve essere indicata in forma di stringa esadecimale che le fa occupare 40 byte.
2. l'elevatissimo numero di GET: l'organizzazione di questo esperimento richiede un numero quadratico di GET rispetto al numero di nodi.

Il secondo punto influisce pesantemente anche sulla durata della simulazione: eseguire un tal numero di query richiede quasi quattro giorni di tempo. L'esecuzione di un numero sufficiente di test avrebbe richiesto un tempo molto elevato.

Il secondo e più importante problema riscontrato nell'utilizzo di Overlay Weaver è relativo al tempo di aggiornamento della finger table dei nodi

utilizzando l'overlay Chord. Nell'articolo che descrive questo overlay[7], si prevede l'utilizzo di un meccanismo che permette di aggiornare le finger table per tenere conto dei nodi che sono entrati a far parte della rete o che l'hanno abbandonata. Questa operazione è molto importante per mantenere la finger table corretta ed efficiente, e permette di evitare che alcuni nodi possano diventare irraggiungibili.

Overlay Weaver implementa in modo poco efficace questa operazione: dopo l'esecuzione di una simulazione della durata di ventidue ore, con intenso scambio di messaggi fra i nodi, soltanto pochi link delle finger table risultano completi. Questo comportamento è stato riscontrato anche in reti molto stabili: la simulazione infatti prevede una prima fase in cui i nodi sono inseriti nella rete, e dopo la quale nella simulazione non avvengono né nuove operazioni di JOIN né operazioni di LEAVE.

Per cui, data l'elevata quantità di tempo a disposizione della rete per stabilizzarsi (fino a 22 ore), è possibile concludere che questo comportamento sia un bug funzionale di Overlay Weaver.

Il problema è stato aggirato attraverso un aggiornamento delle finger table effettuato ad ogni messaggio ricevuto da un altro nodo: nei pacchetti di Overlay Weaver è inclusa un'intestazione che contiene l'ID e l'indirizzo IP del nodo che ha spedito il pacchetto, dati necessari e sufficienti per correggere un eventuale link errato della finger table. Questa soluzione non influisce sull'aggiornamento del link $rt_i$ della finger table che punta ad un nodo che ha abbandonato la rete: l'aggiornamento è effettuato attraverso la ricezione di un messaggio che indica la presenza di un nodo migliore rispetto a quello presente. Un nodo che abbandona la rete non genera messaggi che raggiungono tutti i nodi, per cui in questo caso l'aggiornamento dei link è lasciato al vecchio sistema di aggiornamento di Overlay Weaver.

Non è però stato necessario tenere in conto questo comportamento in quanto esso non si presenta nelle simulazioni effettuate: un nodo che entra a far parte della rete simulata non esegue né un'operazione di LEAVE né ha la possibilità di fallire abbandonando così la rete.

## 5.3 - Simulatore del routing basato sul carico

Lo sviluppo dell'algoritmo di individuazione di un nodo sovraccarico descritto nel prossimo capitolo ha richiesto un buon numero di esecuzioni per controllare e debuggare il codice sorgente. Dover eseguire queste operazioni effettuando una simulazione ad ogni modifica avrebbe rallentato il lavoro eccessivamente a causa dell'elevata durata di una simulazione. L'operazione di debug, inoltre, avrebbe dovuto tener conto anche di quale nodo stava eseguendo il codice di individuazione degli hot spot, complicando ulteriormente l'operazione.

Per evitare questi problemi ho scelto di implementare uno strumento che potesse simulare l'esecuzione di un algoritmo di individuazione di un nodo sovraccarico.

I dati necessari all'esecuzione dell'algoritmo di individuazione degli hot spot sono ottenuti dai risultati dell'esecuzione di una simulazione con Overlay Weaver: le finger table e le corrispondenti hit table sono state raggruppate in due file che indicano il nodo che ha originato le due strutture.

Utilizzando questi dati è possibile applicare l'algoritmo di individuazione degli hot spot senza modificare i dati originali, permettendo così di riutilizzarli sia per le operazioni di debug che per la validazione dell'algoritmo.

Il programma di simulazione del routing basato sul carico, applica l'algoritmo di individuazione degli hot spot sequenzialmente per ogni nodo della rete: nella sua esecuzione considera il nodo analizzato come il nodo che fa partire una richiesta di individuazione degli hot spot, che si conclude con una delle due condizioni:

1. terminazione con successo: uno dei nodi coinvolti nel routing fa parte di un hot spot e si considera sovraccarico
2. terminazione con insuccesso: è stato eseguito un routing che ha attraversato l'intero spazio degli identificatori senza che nessun nodo si considerasse sovraccarico

Alla conclusione dell'analisi su di un nodo, il programma analizza il nodo successivo fino al completamento dei nodi presenti nella rete. Per ogni nodo è registrato in una tabella il risultato dell'analisi, ovvero se è stato trovato un nodo che si considera sovraccarico e quale esso sia. Conoscendo i dati della simulazione è possibile verificare se il comportamento dell'algoritmo è privo di errori, ovvero se i nodi identificati come sovraccarichi lo sono realmente, e la percentuale di successo dell'algoritmo.

L'utilizzo di questo strumento è stato molto utile in quanto l'applicazione di un algoritmo di individuazione degli hot spot ha una durata molto limitata, nel ordine delle decine di secondi. Seppure sia necessario avere a disposizione i dati di una simulazione eseguita con Overlay Weaver i vantaggi sono evidenti: senza l'utilizzo di questo strumento, una simulazione con Overlay Weaver avrebbe fornito i risultati dell'applicazione di un'unica esecuzione di un algoritmo, mentre in questo modo i risultati possono essere riutilizzati più volte anche con versioni differenti dell'algoritmo.

# Capitolo 6 - Individuazione dei nodi sovraccarichi

Le analisi eseguite fino ad ora hanno permesso di stabilire una funzione che approssima il valore atteso $\overline{ht_i}$ della quantità di inoltri effettuati tramite un link della finger table di un nodo: confrontando questo valore con il valore rilevato $ht_i$ è possibile effettuare una stima del livello di carico della sezione dello spazio degli identificatori $si_i$ ad esso collegato.

Ammettendo che la stima sia corretta, non si è comunque in grado di stabilire se in quello spazio degli identificatori vi sia o meno un nodo sovraccarico, in quanto il carico dello hot spot potrebbe essere gestito da un numero sufficiente di nodi.

Lo spazio degli identificatori di un nodo sovraccarico, al contrario, sarà facilmente riconosciuto come un hot spot a causa dell'elevato numero di query che lo raggiunge. Per cui il sistema di individuazione dei nodi sovraccarichi potrà limitarsi a cercare i peer sovraccarichi all'interno degli hot spot.

Il nodo $n_1$ che viene contattato per un'operazione di routing basato sul carico, confronta le stime di carico che associa ai link della finger table, e sceglie quella che considera maggiormente carica. Contattando il nodo $n_2$ corrispondente a quel link, si potrà avere una stima più verosimile del carico di quella sezione dello spazio degli identificatori: le distanze fra i peer in un hot spot bilanciato sono ridotte a causa del numero di nodi in esso contenuti, per cui ognuno dei nodi avrà un buon numero di link che ricadono nello hot spot.

La visione dello hot spot di $n_2$ sarà molto diversa da quella di $n_1$ : mentre quest'ultimo lo considera come un'unica sezione dello spazio degli identificatori, $n_2$ vi sarà connesso con un numero maggiore di link, per cui lo hot spot sarà

considerato come un insieme di diverse sezioni dello spazio degli identificatori ognuna con la sua stima di carico associata. Perciò la stima del carico di $n_2$ sarà approssimata in modo migliore rispetto ad $n_1$, permettendo di evitare l'analisi di sotto sezioni dello hot spot non sovraccariche.

Un sistema di individuazione dei nodi sovraccarichi permette di analizzare esclusivamente le sezioni dello spazio degli identificatori che sono stimate come più cariche, ciò permette di ridurre il numero di hop necessari per esaminare l'intero spazio degli identificatori.

La capacità dell'algoritmo di individuare il nodo sovraccarico dipende dalla bontà delle stime effettuate dai peer che intervengono nel routing, se queste non sono perfette vi è la possibilità che un nodo sovraccarico non sia individuato. La probabilità di individuazione dipende dalla quantità di query che raggiungono il nodo: un numero maggiore di richieste influenza in modo più incisivo la distribuzione delle query sui link della finger table dei nodi che lo precedono.

Per realizzare il processo di individuazione dei nodi sovraccarichi è necessario utilizzare un'operazione distribuita che coinvolge più peer per generare un routing guidato dalle stime di carico dello spazio degli identificatori. Per far ciò è necessario introdurre un nuovo tipo di operazione che chiamerò LBR (load based routing) che, a differenza degli altri messaggi già implementati nelle DHT, non contiene un identificatore che ne definisce la destinazione.

In assenza di una destinazione, il routing non potrà seguire lo schema tipico delle DHT, nel quale ad ogni passo di routing si sceglie il link che minimizza la distanza con l'obiettivo della query. Il link scelto per proseguire l'operazione di routing di un messaggio LBR è quello che il peer corrente stima come più carico. In assenza di un link a cui corrisponde un'elevata quantità di carico, l'operazione di routing inoltra il messaggio verso il primo link la cui stima del carico non è considerata affidabile: la dimensione della sezione dello spazio degli identificatori $si_i$ cresce esponenzialmente all'aumentare dell'indice $i$, ovvero all'aumentare della distanza fra il nodo e $si_i$. In una sezione $si_i$ di ampie dimensioni non è

possibile effettuare una stima accurata del carico: un hot spot potrebbe essere nascosto da sezioni limitrofe dello spazio degli identificatori più scariche.

Per questo motivo è opportuno limitare la distanza massima a cui in messaggio LBR può essere inoltrato, ottenendo un incremento del numero di hop necessari per il completamento di questa operazione.

## 6.1 - L'algoritmo

Il sistema di individuazione dei nodi sovraccarichi descritto è stato implementato utilizzando il seguente algoritmo:

```
n.findSovraccarico(n1,distanza){
    if(n.sovraccarico==true){
        return n;
    }else{
        distanza -= (n.id - n1.id)
        if (distanza <=0){
            return null;
        }else{
            if(htnext>γ){
                next = n.next();
            }else{
                hits = n.calcolaVariazioniPercentuali(HT);
                next = n.indiceMaxVariazione(hits, RT, rtmax);
            }
            return next.findSovraccarico(n,distanza);
        }
    }
}
```

La notazione utilizzata in questo pseudo-codice è in stile RPC, nella quale un'invocazione di funzione è preceduta dal nodo in cui è eseguita. Per esempio, la notazione:

```
next.findSovraccarico(n,distanza);
```

indica l'invocazione della funzione *findSovraccarico* sul nodo *next*.

Le variabili utilizzate nell'algoritmo sono :

- $n$ è il nodo su cui è eseguita la funzione
- $n_1$ è il nodo che ha invocato la funzione,
- *distanza* è un parametro che serve a passare fra le varie invocazioni della funzione la distanza già percorsa dal routing.
- $\gamma$ è il valore di soglia che permette di determinare se un nodo è sovraccarico
- $HT$ è l'intera hit table, la tabella dove sono conservate le osservazioni del traffico che attraversa il nodo,
- $ht_{next}$ è il livello di carico del link al nodo successore.
- $RT$ È la routing table del nodo (finger table),
- $rt_{max}$ è il link che collega il nodo alla sezione più lontana dello spazio degli identificatori a cui si accetta di inoltrare la richiesta di routing.

Il nodo remoto $n_1$ che invoca la funzione su $n$ , passa nei parametri, oltre se stesso, anche la distanza ancora da percorrere per completare un routing dell'intero spazio degli identificatori: nel caso in cui il routing di una richiesta completi un intero giro dello spazio degli identificatori senza trovare un nodo sovraccarico allora la ricerca si considera fallita.

Il primo passo dell'algoritmo è di controllare se il nodo attuale sia sovraccarico: in tal caso la ricerca è stata completata con successo e si notifica il nodo $n_1$ il risultato dell'operazione restituendo il nodo sovraccarico.

Se $n$ non è sovraccarico allora è necessario continuare il routing basato sul carico. Prima di ciò si controlla se l'operazione di routing ha effettuato un'analisi dell'intero spazio degli identificatori. Utilizzando il parametro *distanza* è possibile determinare se l'operazione di routing è conclusa o meno: al valore del parametro, originariamente inizializzato con la dimensione dell'intero spazio degli identificatori, è sottratta la distanza percorsa durante l'ultimo hop del routing. Quando il valore di *distanza* diventa uguale a zero o negativo, allora l'operazione

di LBR ha compiuto un giro dello spazio degli identificatori senza riuscire ad individuare un nodo sovraccarico: questo risultato è considerato un insuccesso che indica l'assenza di un nodo sovraccarico nella rete, o l'incapacità dell'algoritmo di individuarlo.

Se l'operazione di routing non è ancora conclusa, si entra nel cuore dell'algoritmo: utilizzando i dati rilevati dall'analisi del carico si stima quale link della finger table sia il più sovraccarico determinando il collegamento attraverso il quale proseguire il routing del messaggio LBR.

Con *calcolaVariazioniPercentuali( HT )* si stima, per ogni link della finger table, il suo livello di carico. Questa operazione è eseguita calcolando il valore di :

$$\frac{ht_i - \overline{ht_i}}{\overline{ht_i}}$$

che determina la variazione relativa del carico rilevato $ht_i$ rispetto al valore atteso $\overline{ht_i}$ .

Il valore di $\overline{ht_i}$ è calcolato utilizzando la formula:

$$\overline{ht_i} = \begin{cases} b * e^{(c*(i-i_{max}))} + d - b & i \geq i_{max} \\ \dfrac{b_1}{1 + e^{(b_2 - b_3*(i-i_{max}))}} & i < i_{max} \end{cases}$$

come presentato nel capitolo 4.

La stima del carico del link $rt_{next}$ che collega il nodo al suo successore, è trattata in modo particolare: data l'elevata correlazione fra il valore ricavato dall'osservazione del traffico con il reale numero di query che raggiungono il nodo successore, la stima di carico è effettuata confrontando il valore $ht_{next}$ con il valore di $\gamma$ . Se il valore di $ht_{next}$ è maggiore del valore di soglia $\gamma$ allora il nodo successore è considerato sovraccarico e lo si sceglie come prossimo nodo del routing, altrimenti si sceglie il link $rt_i$ a cui corrisponde il valore massimo di *hits* tramite la funzione i*ndiceMaxVariazione*.

L'insieme dei link selezionabili da questa funzione non coincide con l'insieme dei link presenti della finger table: il valore dell'indice $i$ di $rt_i$ di un link selezionabile è limitato dal valore del parametro $rt_{max}$ che indica l'ultimo link di cui la stima del carico è considerata affidabile.

Nel caso in cui nessun link compreso fra $(0, rt_{max})$ presenti un valore di $ht_i > \overline{ht_i} + \epsilon$ con $\epsilon = \frac{\overline{ht_i}}{2}$, e quindi non sia considerato sufficientemente carico dalla funzione *nodoIndiceMaxVariazione* allora il link scelto per proseguire il routing è $rt_{max+1}$, che corrisponde al primo nodo della sezione in cui il nodo non considera affidabile la stima del suo carico. Il valore di $\epsilon$ è stato così scelto in quanto corrisponde alla variazione rilevata da un nodo in un link collegato ad un hot spot pari a 4 volte il carico di un nodo.

## 6.2 - Risultati dei test eseguiti sull'algoritmo

La capacità di rilevare un nodo sovraccarico da parte dell'algoritmo proposto è stata analizzata tramite l'esecuzione di una simulazione: è stata creata una rete con 1023 nodi bilanciati ed un solo nodo sbilanciato. Il test applica l'algoritmo su ogni nodo della rete richiedendo di localizzare il nodo sovraccarico.

Ai nodi della rete sono stati assegnati degli identificatori casuali distribuiti uniformemente sull'intero spazio degli identificatori, in questo modo la rete ha una distribuzione dello stesso tipo di quelle riscontrabili applicando il consistent hashing.

La distribuzione casuale dei nodi all'interno dello spazio logico degli indirizzi causa degli agglomerati di nodi vicini: molti nodi a poca distanza rendono la loro sezione dello spazio degli identificatori uno hot spot. In queste condizioni, l'algoritmo deve individuare il nodo sovraccarico in presenza di hot spot che contengono nodi bilanciati.

L'algoritmo sarà esaminato considerando due fattori:

1. la percentuale di successi: ossia il numero delle operazioni di routing che hanno individuato il nodo sovraccarico
2. il numero di hop necessari per individuare il nodo sovraccarico.

La simulazione è stata creata ed eseguita con Overlay Weaver, allo scopo di acquisire i dati relativi all'operazione di routing, mentre l'algoritmo è stato applicato a questi dati tramite l'utility di cui si è parlato nel capitolo 5.

I passi presenti nel file di scenario di Overlay Weaver sono:

1. inserimento dei nodi: la posizione dei 1023 nodi bilanciati è totalmente casuale, mentre il il nodo sovraccarico è posizionato nell'identificatore 0. Ciò serve a poter confrontare facilmente i dati ricavati da simulazioni diverse, mentre non influisce sulla distribuzione dei nodi.
2. Inserimento delle chiavi: ad ogni nodo è stata assegnata una chiave con il medesimo identificatore del nodo, al nodo sovraccarico sono state

assegnate un numero superiore di chiavi. In base alla simulazione, il numero di chiavi ha assunto i valori di 8, 16, 32.

3. Stabilizzazione delle finger table: ogni nodo ha eseguito una operazione di GET verso tutti gli altri nodi, aggiornando la sua finger table ad ogni messaggio ricevuto.

4. Azzeramento valori: I valori rilevati e registrati nella hit table sono stati cancellati per non influire sull'analisi dei dati

5. Ricerca delle chiavi: ogni nodo ha eseguito un'operazione di GET verso ogni chiave della rete.

La prima analisi effettuata sui dati delle simulazioni eseguite è stata la determinazione della probabilità di successo ottenuta dall'algoritmo: i risultati sono mostrati nel Grafico 26. I valori mostrati in questo grafico sono relativi al numero totale di richieste eseguite in tutte le simulazioni il cui nodo sovraccarico presenta il medesimo fattore di sbilanciamento.

I risultati mostrano una bassa capacità dell'algoritmo di individuare dei nodi con un fattore di sbilanciamento di 8: in questo caso la probabilità che ha una richiesta LBR di individuare il nodo sovraccarico è infatti inferiore al 40% . L'incremento del fattore di sbilanciamento modifica consistentemente questo valore: con un fattore di sbilanciamento di 32, la probabilità di individuare il nodo sovraccarico passa al 82%.

Nelle simulazioni in cui il nodo sovraccarico presenta il medesimo fattore di sbilanciamento, la probabilità di successo dell'algoritmo è una quantità molto variabile: in alcune simulazioni tutti i nodi sono stati in grado di individuare il nodo sovraccarico, mentre in altre è stato individuato soltanto in poche richieste di LBR.

La distribuzione delle simulazioni con un fattore di sbilanciamento di 32 suddivise per numero di operazioni LBR completate con successo è presentata nel Illustrazione 8. Nel 67,5% delle simulazioni tutti i nodi della rete sono stati in grado di individuare con successo il nodo sovraccarico presente nella rete.

Ho analizzato la distribuzione dei nodi che hanno dato origine a richieste di LBR completate senza successo, essa ha una conformazione a “grappoli”: tali peer si presentano in gruppi distribuiti casualmente nello spazio degli identificatori. La distanza fra i nodi di un medesimo gruppo è generalmente piuttosto ridotta.

Analizzando i percorsi della richiesta LBR eseguito dai nodi che appartengono ad un medesimo gruppo, ho riscontrato che il loro routing converge in un unico percorso che termina con un insuccesso.

Purtroppo a causa dell'apparente distribuzione casuale dei gruppi di nodi non sono riuscito ad identificare uno schema che possa permettere di migliorare l'algoritmo di individuazione degli hot spot.

I dati citati fino ad ora sono relativi agli esperimenti che hanno utilizzato un valore di $rt_{max}=154$ . Questo valore determina il link della finger table che collega il nodo alla sezione dello spazio degli identificatori più lontana a cui si permette di inoltrare la richieste di LBR.

Sono state effettuate simulazioni con valori differenti di $rt_{max}$ prendendo in considerazione i valori 154, 155, 156 : questi valori sono stati scelti in modo da corrispondere alla sezione discendente della curva che approssima la distribuzione degli inoltri dei link della finger table.

I risultati, mostrati in Tabella 3, sono stati piuttosto inattesi: pur riducendo la probabilità di successo dell'algoritmo essa varia soltanto in lieve misura.

Le cause di una variazione di così lieve entità è stata attribuita alla formazione di hot spot generati dalla distribuzione casuale dei nodi: essi sono piuttosto frequenti nella rete, per cui sono in grado di influenzare la stima del carico di un link in modo più incisivo rispetto ad un hot spot lontano. Quindi anche se i nodi hanno a disposizione un range maggiore in cui scegliere il link di inoltro per l'operazione LBR, essa è utilizzata soltanto raramente.

Il valore di $rt_{max}$ si è dimostrato molto rilevante per un altro parametro del routing: il numero di hop utilizzati per individuare il nodo sovraccarico ha presentato una distribuzione molto differente nei 3 casi analizzati.

Come mostrato nel Grafico 27, che presenta la distribuzione del numero di hop necessari al LBR per individuare il nodo sovraccarico, i dati fra le tre distribuzioni esaminate è molto variabile. La distribuzione corrispondente a $rt_{max}=156$ presenta come classe modale quella corrispondente a 21-25 hop, mentre la classe modale nella distribuzione $rt_{max}=154$ corrisponde a 11-15.

Una tale differenza è dovuta al fatto che anche l'utilizzo di un unico link più lontano può ridurre considerevolmente la distanza di routing: esso può permettere di evitare l'analisi di un agglomerato di nodi, riducendo così il numero di hop necessari per tutte le operazioni LBR che sono inoltrate attraverso quel link.

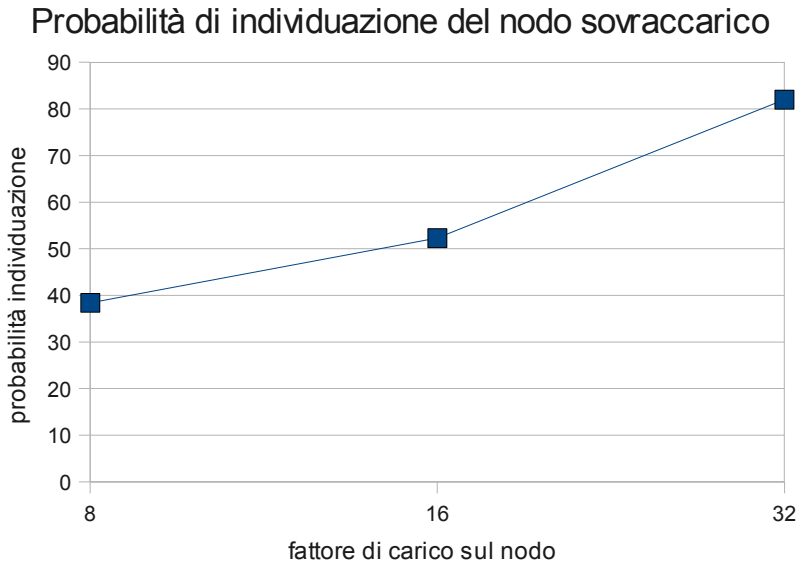


*Grafico 26: Probabilità di individuazione del nodo sovraccarico a variare del fatto di sbilanciamento del nodo*

| Rtmax | 154 | 155 | 156 |
|---|---|---|---|
| Successi % | 82,00% | 79,80% | 77,90% |

*Tabella 3: Percentuale di successi al variare del valore di* $rt_{max}$ *. A percentuali di successo maggiori corrispondono routing con un maggiore numero di hop*


*Illustrazione 8: Ogni valore del grafico rappresenta la percentuale di simulazioni che hanno presentato un numero di ricerche terminate con successo compreso nel range indicato dalla legenda*

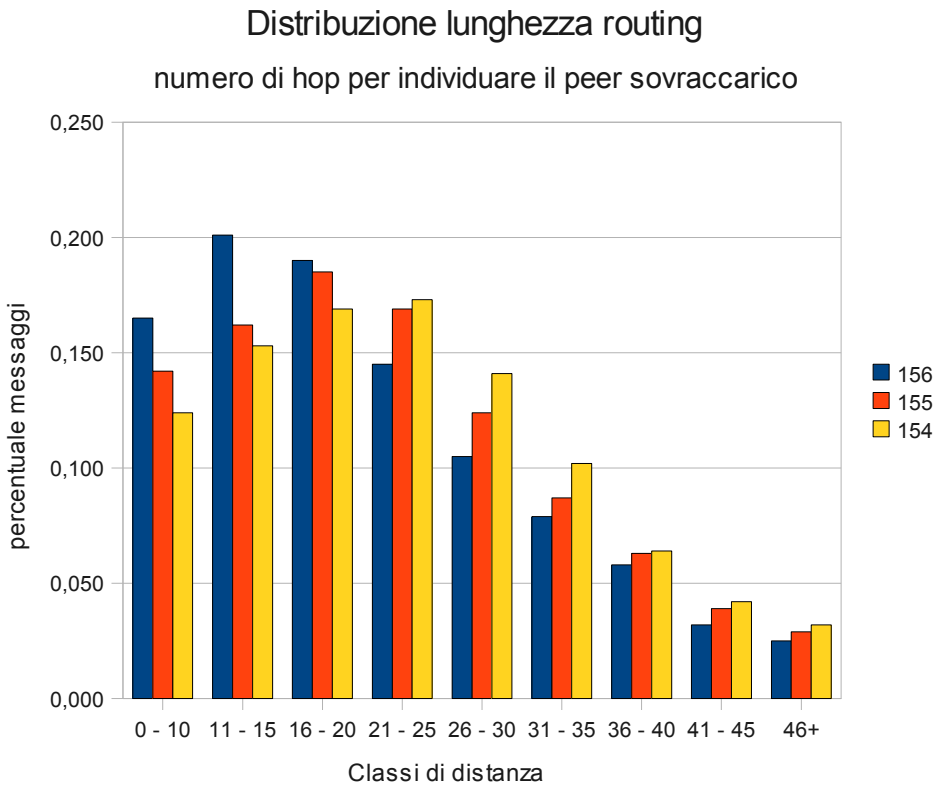


*Grafico 27: Numero di hop necessari per l'individuazione del peer sovraccarico. Si mostrano le differenze al variare di* $rt_{max}$

# Capitolo 7 - Conclusioni

L'obiettivo di questo lavoro di tesi è stata la definizione di un sistema di individuazione dei nodi sovraccarichi di una distributed hash table.

Il sistema si basa su una stima dinamica del carico della rete eseguita attraverso l'analisi del traffico locale ad un nodo. Si è mostrato come la distribuzione sui link della finger table delle query che attraversano un peer sia dipendente dalla quantità di carico delle sezioni dello spazio degli identificatori a cui il nodo è connesso.

Attraverso un analisi di regressione non lineare sui dati raccolti con apposite simulazioni, è stata estrapolata una funzione che permette di stimare il numero medio di query oltre il quale un link è da considerarsi sovraccarico.

È stato mostrato come confrontando il numero di query inoltrate attraverso un link ed il corrispondente valore medio stimato, è possibile determinare il livello di carico di una sezione dello spazio degli identificatori con una precisione che cresce al diminuire della distanza con il nodo.

Infine, è stato proposto un algoritmo, che utilizzando un routing guidato dalle stime del carico eseguite dai nodi (operazione LBR), è in grado di individuare un nodo sovraccarico presente nella rete.

Le analisi eseguite sull'algoritmo hanno mostrato una maggiore efficienza nell'individuare il nodo sovraccarico nel caso in cui il fattore di sbilanciamento sia consistente. Una tale situazione si può verificare più frequentemente in un sistema che modifica la funzione utilizzata nel consistent hashing per mantenere la località delle chiavi.

Il numero di hop necessari all'operazione LBR, è abbastanza limitato: la moda nel caso peggiore fra quelli analizzati ricade nella classe 21-25 con una media di circa 23 hop.

L'algoritmo è stato analizzato variando la dimensione della sezione dello spazio degli identificatori di cui i nodi ritengono la stima del carico affidabile. Da questo parametro è dipendente la distanza massima che può essere percorsa durante un hop.

Con questa analisi si è messo individuato un trade-off fra il numero di hop necessari per localizzare un nodo sovraccarico e la probabilità di individuarlo: incrementando la distanza massima percorribile diminuisce la probabilità di individuare il nodo sovraccarico.

## 7.1 - Sviluppi futuri

Il sistema di individuazione dei nodi sovraccarichi è stato pensato per essere utilizzato in un sistema di bilanciamento del carico di lavoro dei nodi di una DHT che si basa sull'inserimento dei nodi che richiedono di unirsi alla rete direttamente dove il carico è maggiore. In questo modo, la ripartizione dello spazio degli identificatori che avviene durante la funzione di JOIN, è utilizzata per alleviare il carico di un nodo sovraccarico.

In un tale sistema c'è da considerare che è possibile che il churn rate sia inferiore alla velocità con cui nuove chiavi entrino ed escano dalla rete: ciò potrebbe causare l'incapacità del sistema di bilanciamento del carico ad adattarsi tempestivamente alle modifiche della distribuzione delle query nello spazio degli identificatori.

Per questo motivo si potrebbe accostare a tale sistema una tecnica per incrementare il numero di operazioni di JOIN eseguite dai peer. Ciò potrebbe essere realizzato tramite l'utilizzo di una variante dei virtual server [14] nel quale ogni nodo fisico gestisce un numero variabile di nodi logici.

# Bibliografia:


[1], KUROSE, J.; ROSS, K.: «{Internet e reti di calcolatori}», *III edizione Pearson/Addison Wesley*, 2005.

[2] Napster home page. http://www.napster.com.

[3], J. BUFORD, H. YU, E.K.LUA, MORGAN KAUFMANN: *P2P NETWORKING AND APPLICATIONS*, Elsevier, 2008.

[4], POUWELSE, J.; GARBACKI,P.; EPEMA,D.; SIPS, H.: «{The bittorrent p2p file-sharing system: Measurements and analysis}», *Lecture notes in computer science*, 3640, 2005.

[5], ZHAO, B.; KUBIATOWICZ,J.; JOSEPH, A.: «{Tapestry: An infrastructure for fault-tolerant wide-area location and routing}», *Computer*, 74, 2001.

[6], ROWSTRON, A.; DRUSCHEL, P.: «{Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems}», *Lecture Notes in Computer Science*, 2001.

[7], STOICA, I.; MORRIS,R.; KARGER,D.; KAASHOEK,M.; BALAKRISHNAN, H.:*{Chord: A scalable peer-to-peer lookup service for internet applications}*, «Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications», 2001 (inédito).

[8], RATNASAMY, S.; FRANCIS,P.; HANDLEY,M.; KARP,R.; SCHENKER, S.:*{A scalable content-addressable network}*, «Proceedings of the 2001 SIGCOMM conference», 2001 (inédito).

[9], KULBAK, Y.; BICKSON, D.: «{The emule protocol specification}», *eMule project, http://sourceforge. net*, 2005.

[10], KARGER, D.; LEHMAN,E.; LEIGHTON,T.; LEVINE,M.; LEWIN,D.; PANIGRAHY, R.: *{Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web}*, «ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING», 1997 (inédito).

[11], WANG, X.; LOGUINOV, D.: «{Load-balancing performance of consistent hashing: asymptotic analysis of random node join}», *IEEE/ACM Transactions on Networking (TON)*, 15, 2007.

[12], BYERS, J.; CONSIDINE,J.; MITZENMACHER, M.: «{Simple load balancing for

distributed hash tables}», *Lecture notes in computer science*, 2003.
[13], Byers, J.; Considine,J.; Mitzenmacher, M.:*{Geometric generalizations of the power of two choices}*, «Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures», 2004 (inédito).
[14], Rao, A.; Lakshminarayanan,K.; Surana,S.; Karp,R.; Stoica, I.: «{Load balancing in structured p2p systems}», *Lecture notes in computer science*, , 2003.
[15], Roussopoulos, M.; Baker, M.: «{Practical load balancing for content requests in peer-to-peer networks}», *Distributed Computing*, 18, 2006.
[16], Cai, M.; Frank,M.; Chen,J.; Szekely, P.: «{Maan: A multi-attribute addressable network for grid information services}», *Journal of Grid Computing*, 2, 2004.
[17], Schmidt, C.; Parashar, M.: «{Squid: Enabling search in DHT-based systems}», *Journal of Parallel and Distributed Computing*, 2008.
[18] Gnutella - A Protocol for a Revolution. http://rfc-gnutella.sourceforge.net/.
David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, Rina Panigrahy: «*Consistent Hashing and Random Trees:Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*», en , *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997.
[20], El-Ansary, S.; Alima,L.; Brand,P.; Haridi, S.: «{Efficient broadcast in structured p2p networks}», *Lecture Notes in Computer Science*, 2003.
[21] Covarianza. http://it.wikipedia.org/wiki/Covarianza.
[25], Rice, J.: *{Mathematical statistics and data analysis}*, Duxbury press Belmont, CA, 1995.
[22] e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc43.htm.
[23], Rhea, S.; Geels,D.; Roscoe,T.; Kubiatowicz, J.: *{Handling churn in a DHT}*, Computer Science Division, University of California, 2003.
[24], Shudo, K.; Tanaka,Y.; Sekiguchi, S.: «{Overlay Weaver: An overlay construction toolkit}», *Computer Communications*, 31, 2008.
[26], Neter; Wasserman; Kutner; others: *{Applied linear statistical models}*, Irwin Homewood, 1990.

[27] Datafit webpage. http://www.oakdaleengr.com/.

[28] Regression analysis. http://en.wikipedia.org/wiki/Regression_analysis.

[29], LEVENBERG, K.: «{A method for the solution of certain non-linear problems in least squares}», *Q. Appl. Math*, 2, 1944.

[30], MARQUARDT, D.: «{An algorithm for least-squares estimation of nonlinear parameters}», *Journal of the Society for Industrial and Applied Mathematics*, 1963.

[31] Least squares. http://en.wikipedia.org/wiki/Least_squares.

[32] Nonlinear Regression Modeling: Dataset rat42.
http://www.itl.nist.gov/div898/strd/nls/data/ratkowsky2.shtml

[33] Bamboo DHT homepage. http://bamboo-dht.org/.

[34] Overlay Weaver home page. http://overlayweaver.sourceforge.net/.

[35], WELSH, M.: «{The staged event-driven architecture for highly-concurrent server applications}», *University of California, Berkeley*, 2000.

[36] The Bamboo Events Model.
http://bamboo-dht.org/programmers-guide.html.

[37], MAZIERES, D.: «{Self-certifying file system}», 2000.

[38], RHEA, S.; CHUN,B.; KUBIATOWICZ,J.; SHENKER, S.:*{Fixing the embarrassing slowness of OpenDHT on PlanetLab}*, «Proc. WORLDS», 2005 (inédito).

# Ringraziamenti

*La gioia di aver portato a termine questa "Odissea" è enorme.*

*Ringrazio  la mia famiglia per i loro continui incoraggiamenti.*

*Ringrazio Roberto, Livio e Matteo, per la loro continua presenza durante tutto il periodo universitario.*

*Ringrazio tutto "Il Parcheggione" perché siete i migliori amici che si possano desiderare.*

*Ringrazio Mick, per avermi aiutato a districarmi nelle analisi statistiche.*

*Ringrazio Stefano e Riccardo, perché il "Trio Siculo" è stata un'esperienza indimenticabile.*

*Ed in fine ringrazio Giorgia, che ha sopportato i sempre più frequenti "scleri" avuti durante il periodo di tesi.*

*Grazie a TUTTI!!!*