P2P

OVERLAY NETWORKS

Panoramica

Sistemi P2P

- Rete a livello applicativo
- Nodi sono uguali
- Ingresso/uscita di nodi
- Aggiunta/rimozione risorse
- Servizi «emergenti»

Feature	Performance
Auto-organizzazione	Storage, CPU etc. collettive
Controllo decentralizzato	Localizzazione di nodi/risorse veloce
Simmetria dei ruoli	Scalabilità
Anonimato	Gestione di ingresso/uscita (churn)
Naming	Geo-awareness
Secutiry, authentication, trust	Ridondanza (rete, storage, risorse, etc)

Middleware P2P

Implementano le funzionalità base per sistemi P2P:

- Localizzazione e interazione con le risorse
- Inserimento/rimozione risorse
- Inserimento/rimozione nodi

Requisiti non-funzionali:

- Elevata scalabilità
  - Numero di nodi, risorse, carico
- Scala globale
- Bilanciamento del carico
- Ottimizzazione delle interazioni locali
- Disponibilità variabile di nodi/risorse

Indicizzazione dei dati

Dati/risorse identificati mediante indicizzazione

- Indice centralizzato e.g. Napster, Maze, DNS
- Indice decentralizzato:
  - Informazione distribuita tra i peer
  - Protocolli di accesso/reperimento e.g. DHT
  - Differenti tipi di hashing, ricerca, costo, resistenza a fallimenti e churn
- Indice locale:
  - Ogni peer indicizza solo le risorse locali
  - Risorse non locali vanno cercate di volta in volta
  - (unstructured p2p, flooding, random walk)

Indicizzazione semantica (human readable)

- Supporto a ricerche complesse (e.g. attribute based)

Indicizzazione non-semantica (e.g. hashing)

Overlay strutturati e non

Ricerca data-centrica e non host-centrica (cf. WWW)

Gestione e ricerca delle risorse sono intrinsecamente legate alla tipologia di overlay e indicizzazione

Unstrctured P2P overlay:

- Linee guida su ricerca, storage e organizzazione
- Metodi di ricerca principalmente basati su flooding o random-walk
- Topologie emergenti, ottimizzazione locale, rischio frammentazione

Structured P2P overlay:

- Topologie specifiche e.g anelli, ipercubi, mesh
- Protocolli e principi di organizzazione fissati

DHT

Nodi e oggetti sono mappati in uno spazio di chiavi comune

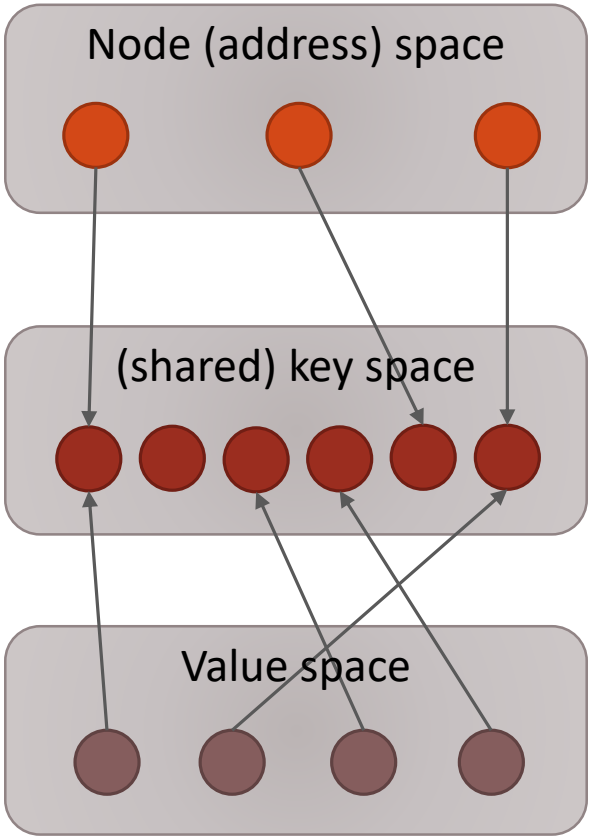
Posizionamento deterministico

Costo di mantenimento degli invarianti

- Join/Leave
- Put/Remove

Lookup veloce

Ricerca per attributi, keywords, gruppi molto complessa o impossibile



Strutturati *vs.* Non-Strutturati

OVERLAY STRUTTURATI

Piazzamento delle risorse deterministico

Lookup veloci

Modifiche più complesse

Hashing basato su caratteristiche singole

Range, keyword, attribute query complesse da supportare

OVERLAY NON-STRUTTURATI

Join/Leave hanno impatto ridotto

Indicizzazione locale

Lookup costosi e/o non certi

- Flloding/random walk
- Best effort

Query complesse (attributi, range)

Topologie emergenti

- Simple Random Graphs
- Power Law Random Graphs

Gnutella

RANDOM TOPOLOGY

Gnutella

Overlay logico non strutturato

- Indicizzazione locale dei contenuti
- Topologia arbitraria
- Ricerca per flooding
- «servent» server + client

Join: formare una connessione con un nodo qualsiasi

- Può essere letta come «merge»

Messaggi:

- *Ping* annuncio nuovi host
- *Pong* porta e statistice del nodo che risponde
- *Query* stringa da cercare, caratteristiche dell'host (e.g. download speed)
- *QueryHit* IP:port, statistiche, metadati, segue il percorso della query.

Ricerca

Assunzioni

- n nodi
- m oggetti
- q_i popolarità dell'oggetti i-esimo ($\sum q_i = 1$)
  - Uniforme: $q_i = \frac{1}{m}$
  - Zipf-like power law: $q_i \propto i^{-\alpha}$ (più realistico)

Replicazione

- r_i repliche dell'oggetti i-esimo ($\sum r_i = R$)
- Uniforme: $r_i = \frac{R}{m}$
- Proporzionale: $r_i \propto q_i$
- Quadratica: $r_i \propto \sqrt{q_i}$

Ricerca

Metriche di interesse per la ricerca di un oggetto:

- Probabilità di successo
- Ritardo o distanza (hops)
- Numero di messaggi processato da ogni nodo
- Copertura (frazione dei nodi visitati)
- Duplicazione dei messaggi
- Massimo numero di messaggi (per nodo)
- Numero di oggetti per criterio (recall) cf. query complesse
- Rapporto tra recall e numero di messaggi

Flooding *vs.* random walk

Flooding:

- Identificatori per evitare cicli
- Diametro di ricerca (TTL o HTL) incrementale

Random walk:

- Identificatori per evitare cicli
- Numero costante (massimo) di messaggi attivi (walkers)

Efficienza:

- Rapporto di recall e diametro di ricerca
- Rapporto tra messaggi e diametro di ricerca
- Contestualizzare con strategie di replicazione

Chaching delle ricerche a cui un nodo ha partecipato (cf switch)

Strategie di replicazione

n, m	Numero di noti e di oggetti nel sistema
q_i	Popolarità dell'oggetti i-esimo ($\sum q_i = 1$)
r_i	Repliche dell'oggetti i-esimo ($\sum r_i = R$)
ρ	Capacità (in termini di numero di oggetti) per nodo ($R = n \cdot \rho$)
A, A_i	Dimensione di ricerca media per tutti gli oggetto e per i
U_i	Utilizzo di ogni replica dell'oggetto i-esimo

	r_i	A	$A_i = {}^{n}/_{i}$	$U_i = \frac{R \cdot q_i}{r_i}$
Unif.	$c, R/m$	m/ρ	m/ρ	$q_i \cdot m$
Prop.	$q_i \cdot R$	m/ρ	$1/(\rho \cdot q_i)$	1
Quad.	$\frac{R \cdot \sqrt{q_i}}{\sum_j \sqrt{q_j}}$	$\frac{\left(\sum \sqrt{q_i}\right)^2}{\rho}$	$\frac{\sum_j \sqrt{q_j}}{\rho \cdot \sqrt{q_i}}$	$\sqrt{q_i} \cdot \sum_j \sqrt{q_j}$

Strategie di replicazione

Assumiamo che le repliche invecchino e siano rimosse

- Rimozione indipendente da utilizzo o popolarità (i.e. nessuna assunzione churn)
- Casuale o FIFO, non LRU

Replicazione

- Uniforme: ogni risorsa è replicata al momento dell'inserimento
- Proporzionale: ogni utilizzo/ricerca crea una copia (e.g. file sharing)
- Quadratico: crea una copia in ogni nodo attraversato da una ricerca

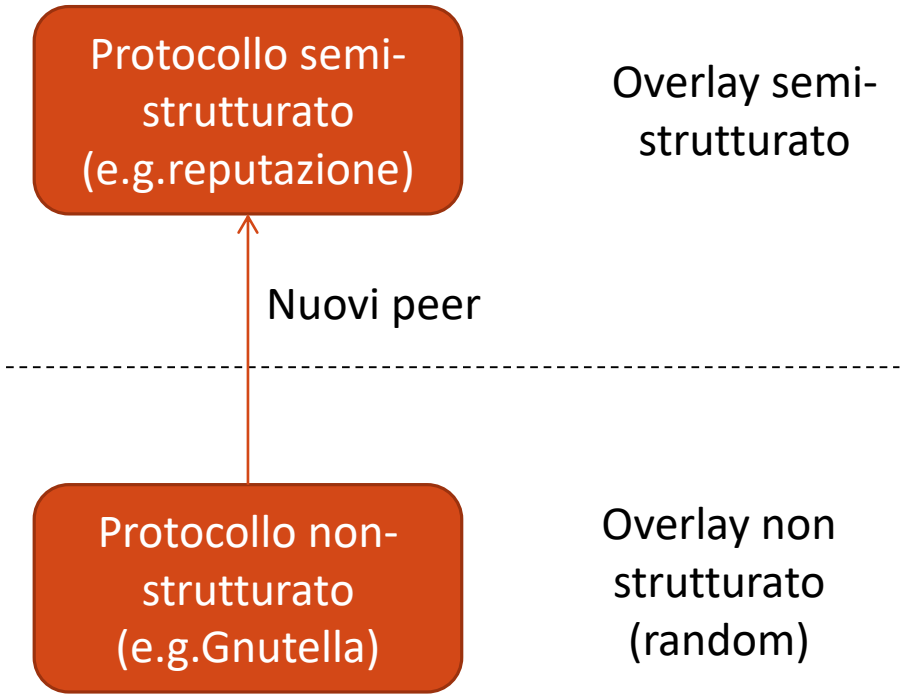
Euristiche vicinato semantico

Filtra i peer del vicinato:

- Distanza
- Contenuti
- Risorse
- Reputazione

Condivide informazioni sui vicini con i vicini

- Push: segnala al vicinato i propri k migliori vicini
- Pull: richiede a un vicino i suoi k migliori vicini



Chord

RING TOPOLOGY

Chord

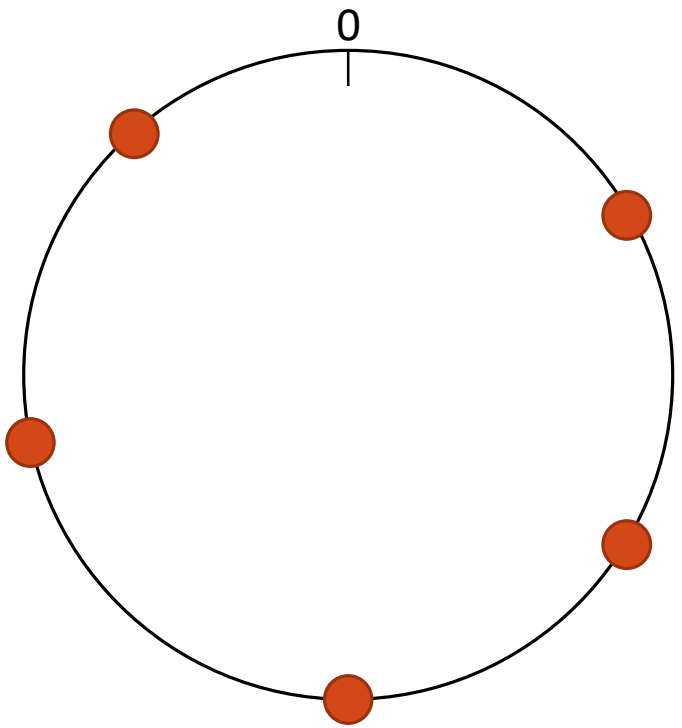
Nodi e oggetti sono mappati in uno spazio di chiavi piatto

Chiavi m-bit

Logicamente organizzato ad anello

Ripartito tra i nodi del sistema

Ogni chiave k è gestita dal primo nodo avente chiave >= k



Lookup

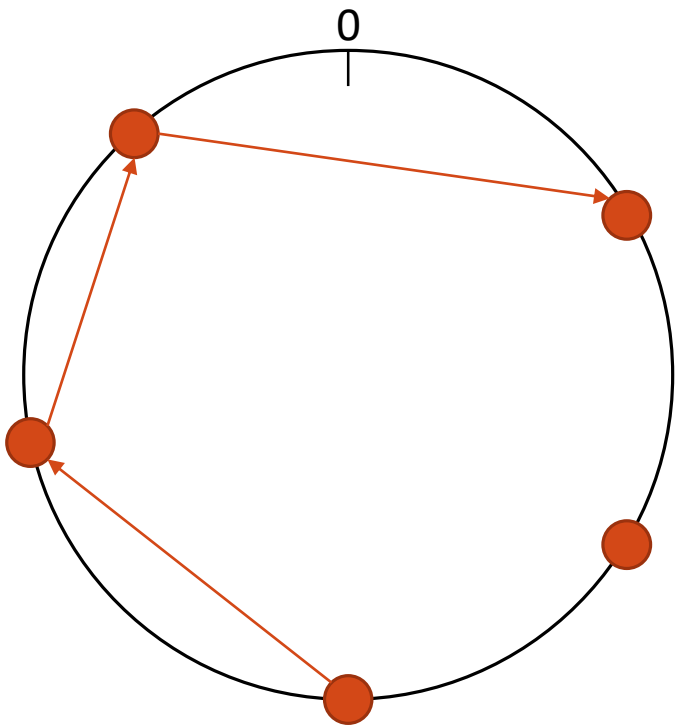
Ogni nodo tiene traccia del proprio
- Successore
- Predecessore

Una richiesta di lookup è instradata lungo l'anello

```
n.lookup(key):
  if key in (n.key,
n.succ.key]
    return n.succ
  else
    return n.succ.lookup(key)
```

Spazio: O(1)

Messaggi: O(N)



Churn

```
n.join_ring(bootstrap) // semplificazione n.key è univoca
  n.pred = null
  n.succ = bootstrap.lookup(n.key)

n.stabilize() // eseguita periodicamente
  m = n.succ.pred
  if n != m
    n.succ = m
  n.succ.notify(n)

n.notify(m) // m ritiene m.succ = n
  if n.pred = null or m.key in (n.pred, n.key]
    key_handover(m)
    n.pred = m
```

Lookup scalabile

Ogni nodo mantiene una tabella di instradamento (**finger table**)

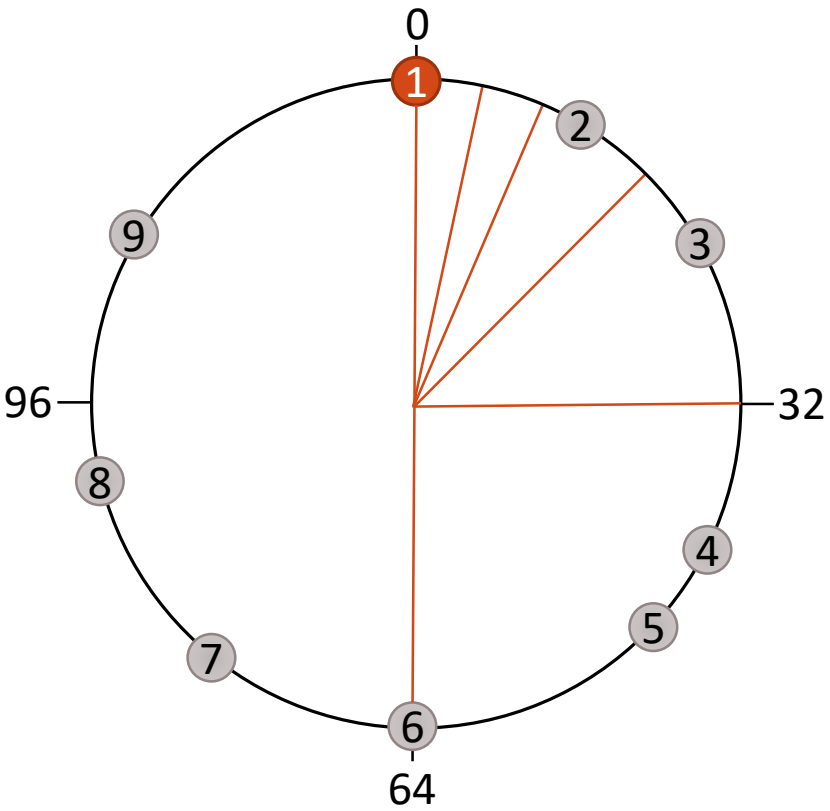
- Coppie chiave -> nodo
- O(log(N)) coppie
- Le chiavi sono distribuite per incrementi esponenziali della chiave del nodo
- A ogni chiave è associato il nodo che la gestisce

Key	Node
$k+1$	n.succ
$k+2$	…
$k+4$	
$k+8$	
$k+16$	
…	
$k+2^{m-1}$	

Aggiornamento periodico:

```
for i in [0, m)
  finger[i]=lookup(key + 2^(m-1))
```

Lookup scalabile



Key	Node
$k+1$	2
$k+2$	2
$k+4$	2
$k+8$	2
$k+16$	3
$k+32$	4
$k+64$	6

Lookup scalabile

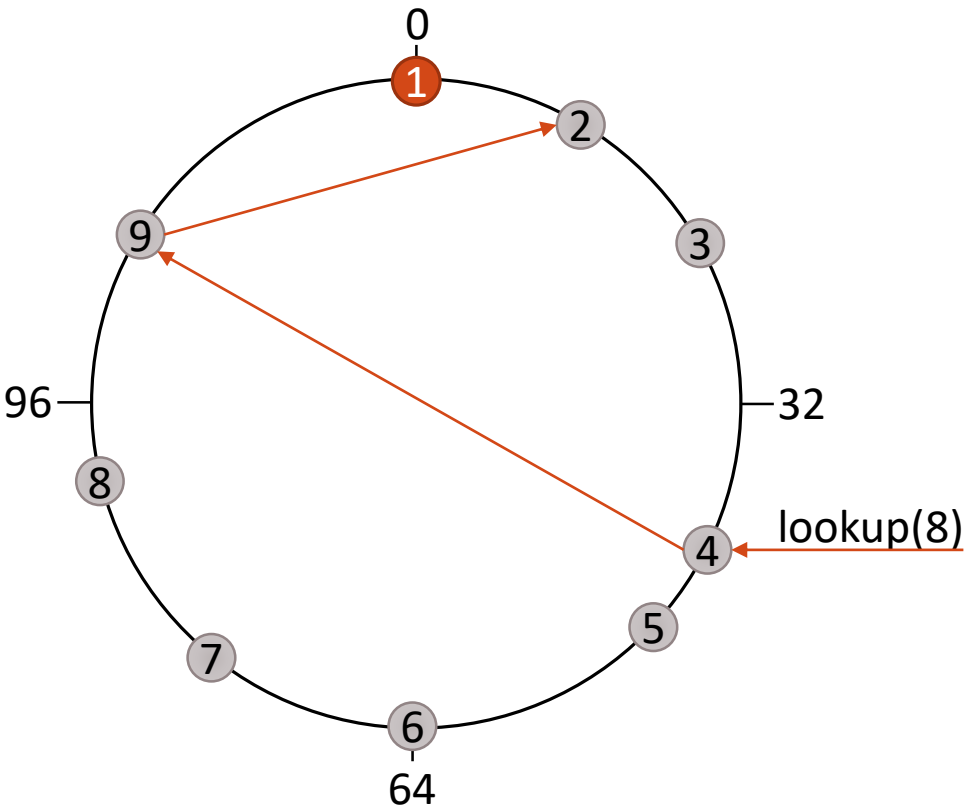
```
n.lookup(key):
  if key in (n.pred.key, n.key]
    return n
  else
    s = n.closestPred(key)
    return s.lookup(key)

n.closestPred(key):
  for i = m-1 down to 0
    if finger[i] not in (n.key, key]
      return finger[i]
```

Maggiore precisione per le chiavi «vicine»

Il lookup procede con salti via via più precisi

Lookup scalabile



Node	Key
1	0
2	10
3	20
4	39
5	45
6	64
7	77
8	90
9	105

Resistenza ai fallimenti

Impatto dei fallimenti

K successori

- K fallimenti (quasi) contemporanei di nodi adiacenti

Ring crawl

- Sfrutta la finger table per ripristinare l'anello

Replicazione

- Disseminazione delle repliche
- Consistenza
- Heart-beat/leasing/republish

Chord

Dato un anello con chiavi di m di N nodi

- È altamente probabile che ogni nodo abbia in carico al più $^{(1+\varepsilon)m}/_{N}$
- ε è limitato da O(log N)
- È altamente probabile che lookup() richieda O(log N) RPC
- Il tempo di lookup medio è ½ log(N)

Bilanciamento del carico

- Disaccoppiamento risorse e risoluzione
- Key-stealing

Chord-gerarchici e reto non-omogenee

CAN

HYPER-THORUS TOPOLOGY

Content Addressable Network

DHT con spazio delle chiavi iper-toroidale

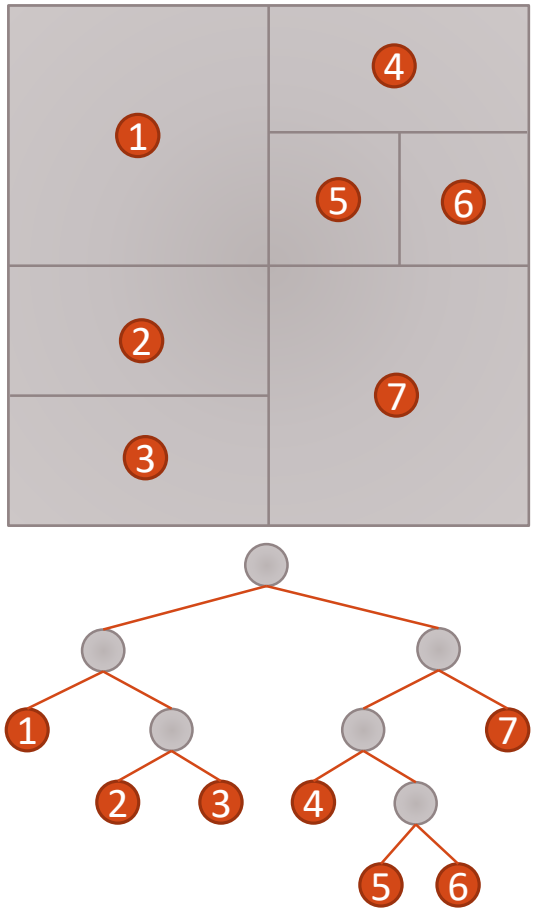
Il d-toroide è ripartito tra i nodi

Join:

- Il nuovo nodo n genera un punto p
- contatta m = bootstrap.lookup(p)
- m divide la propria regione con n
- n si notifica al vicinato

(lo split prevede un ordine tra le dimensioni)

Il vicinato coinvolto dalle operazioni di churn è limitato dalle dimensioni O(d)



CAN maintenance

Heart-beat periodico verso il vicinato

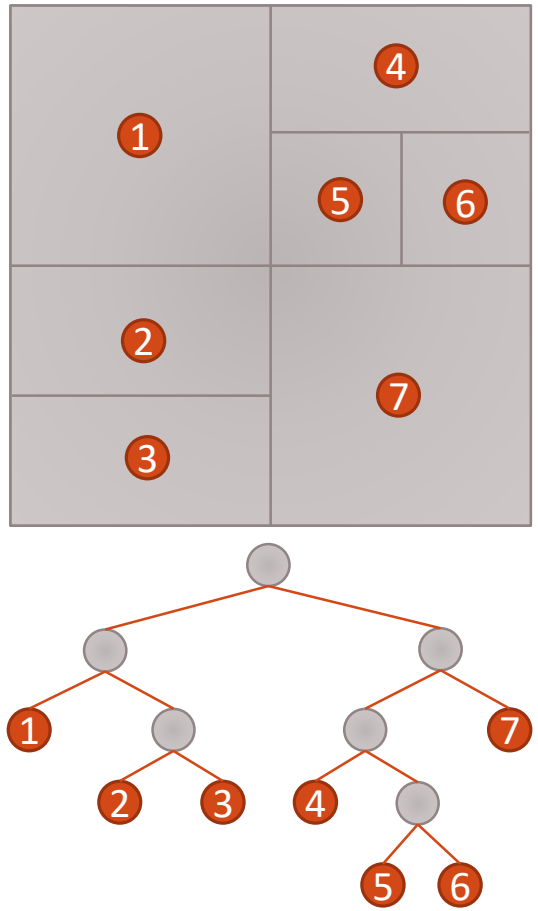
- (cf notify in chord)

Bilanciamento periodico dell'albero

Lo stesso meccanismo è alla base della gestione di fallimenti e uscite

E.g.:

- Se 2 fallisce la sua regione è assegnata a 3
- Se 7 fallisce 5 e 6 sono fuse e 5 prende in carico la regione di 7



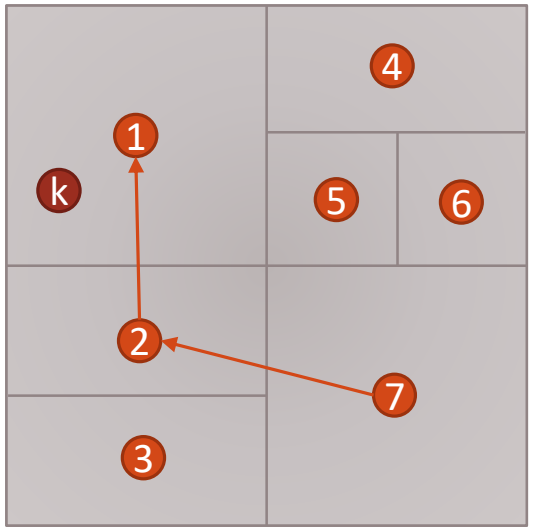
Routing

Ogni nodo traccia i propri vicini (e le loro regioni)

Ogni nodo instrada le richieste in modo greedy al vicino più prossimo al punto da cercare.

La dimensione dello stato mantenuto dai singoli nodi è indipendente da N (cf Chord)

Lunghezza media del percorso $^{d}/_{4} \cdot N^{1/d}$ (cf Chord)



Ottimizzazioni

Dimensioni maggiori

- Velocizza il lookup
- Aumenta la resistenza ai fallimenti
- Aumenta spazio e traffico per mantenere l'infrastruttura

Realtà multiple

- Usa più spazi di coordinate
- Ogni nodo ha regioni diverse nei vari spazi
- Risorse replicate
- Routing salta tra le realtà

Differenti metriche

- Ritardi, posizione geografica …

Tapestry

PREFIX ROUTING

Tapestry

Spazio di indirizzamento A^m

Distanza per prefisso: $d(s,t) = \max\{|p| \mid ps' = s, pt' = t\}$

Una chiave s è gestita dal nodo n più vicino:

- $d(s,n) = \min_m\{d(s,m)\}$

Routing:

- Ogni nodo n mantiene una tabella $m \times |A|$
  - La riga i-esima è associata al prefisso comune con n di lunghezza i p_i
  - Ogni cella (i,a) contiene un nodo con prefisso "p_ia" o il nodo noto più vicino ad esso
- Ad ogni richiesta per s, n
  - Seleziona la cella associata con il prefisso più lungo in comune con s
  - Se la cella contiene n serve la richiesta altrimenti la instrada

Kademlia

XOR ROUTING

Kademlia

Servizio di storage e lookup

Usato in molti sistemi di file sharing

- eMule
- Trackerless BitTorrent

Obbiettivi

- Minimizzare il traffico di mantenimento
- Informazione sulla struttura è diffusa come side-effect dei lookup
- Routing privilegia i link a bassa latenza
- Traffico simmetrico (cf Chord)
- (Richieste parallele/asincrone per ridurre l'impatto di fallimenti)
- (…)

Spazio delle chiavi

Spazio di indirizzamento a m-bit (solitamente 160)

Condiviso tra nodi e risorse

Le risorse sono mantenute dal nodo più vicino rispetto alla metrica XOR

La distanza d(i,j) è data dallo XOR delle stringhe di bit per i e j visto come intero

- E.g. d(1011,0111) = 1100 = 24

Unidirezionalità ($\forall x, \forall \delta > 0 \; \exists! \, y \, . d(x,y) = \delta$) e simmetria $d(x,y) = d(y,x)$

- Lookup per la stessa chiave convergono lungo lo stesso percorso
- Caching lungo tale percorso
- Topologia simmetrica

Routing table

Ogni nodo mantiene una tabella $m \times k$ (impl. incrementale)

La i-esima riga (k-bucket) contiene (al più) k nodi a distanza in $[2^i, 2^{i+1})$

La dimensione dei bucket tende a salire col crescere di i

K è scelto in modo che il fallimento di k (e.g. 20) nodi in un dato intervallo (e.g. un ora) sia sufficientemente poco probabile

I nodi nelle righe sono ordinati per «freschezza»

Quelli contattati più di recente in fondo (cerco di contattare i più vecchi)

Aggiornamento sia dal traffico in entrata che quello in uscita

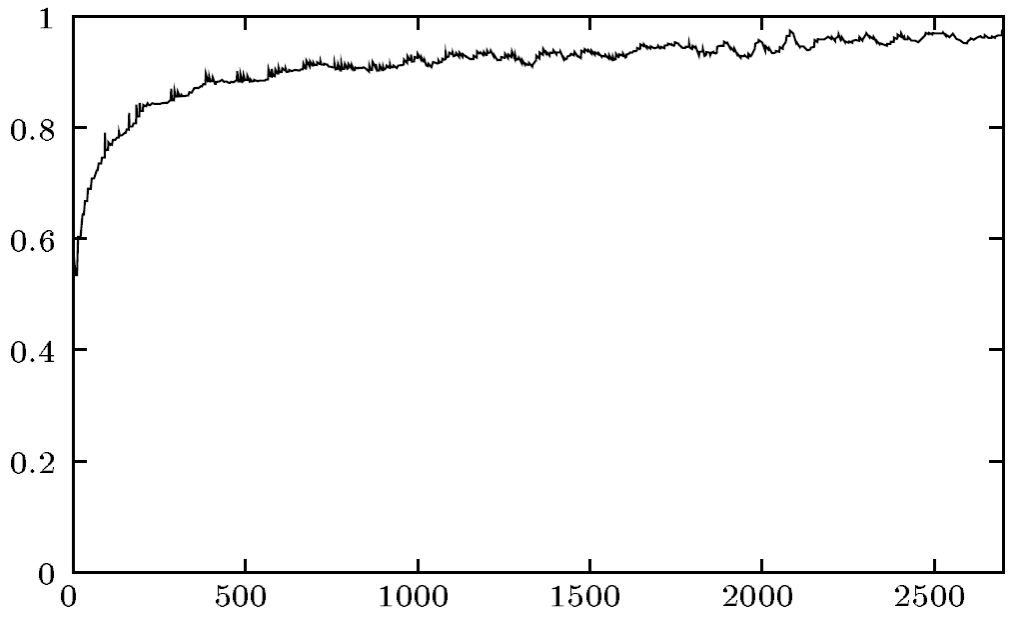
Quando viene scoperto un nuovo nodo è aggiunto in testa al bucket:

- Se non è pieno
- Se pieno e il nodo di testa (meno fresco) non risponde ai tentativi di ping

Routing table

I nodi meno recenti sono privilegiati

Studi su altre reti (e.g. Gnutella) hanno rilevato che la probabilità di permanenza nella rete aumenta «all'invecchiare» del partecipante



Frazione di nodi che sopravvive almeno x minuti dopo una permanenza di almeno x+60 minuti

Lookup

Kademlia offre due primitive di ricerca: una per i nodi e una per i valori

n.find_node(key)

- n se n.key = key, altrimenti
- i k nodi più prossimi a key conosciuti da n

n.find_value(key)

- come sopra **tranne** quando n mantiene un valore val per key
- (i.e. è stata invocata n.store(key,val) nell'intervallo di leasing)

Il lookup avviene per «raffinamento» invocando find_node in parallelo

Lookup

E.g. n ricerca p

1. n computa s = d(n.key, p)
2. Seleziona gli r nodi più vicini a p che conosce e invoca find_node(p)
3. (Aggiorna i k-bucket)
4. Se p non è tra i risultati n ripete 2 con i nuovi nodi (termina se non ve ne sono)

La procedura determina il nodo associato a p oppure i k-nodi più prossimi

La ricerca dei valori usa find_value al posto di find_node terminando non appena un valore è restituito o non vi sono più nodi nuovi

Store e cache

Per inserire la coppia (key,val) si localizzano i k nodi più prossimi a key

Il valore val è salvato su ognuno di essi

I valori sono mantenuti per un dato intervallo (leasing)

Deve essere rinnovato in modo esplicito (re-publish)

Non è prevista cancellazione esplicita

Semplifica l'implementazione di cache

- Ogni nodo attraversato da find_value ne registra l'esito
- Il leasing nelle cache decresce in modo esponenziale rispetto alla distanza

(soluzione simile a tapestry)

Churn

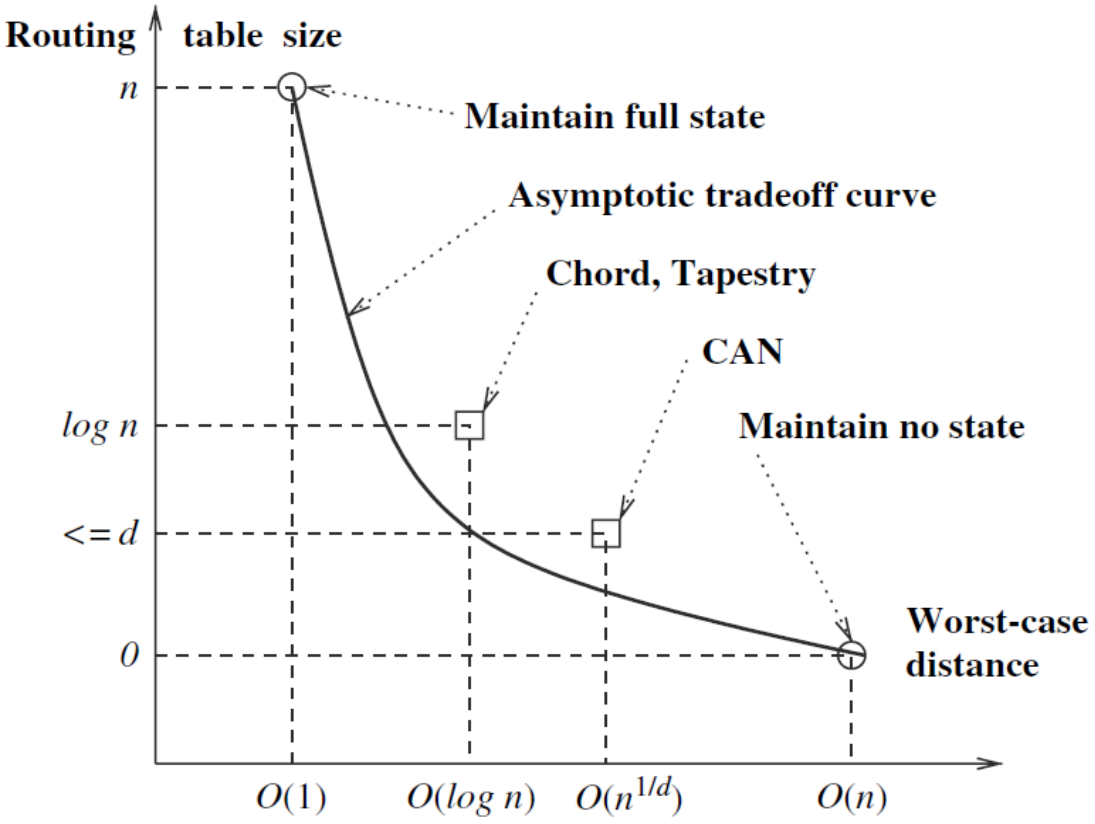
Aggiornamento tabelle:

- Dal traffico
- Lookup periodico
  - Si seleziona una chiave casuale nel bucket da aggiornare
- (scambio di peer coi vicini)

Inserimento di nuovi nodi:

- Necessita di un nodo di bootstrap
- Il nuovo nodo n inserisce il bootstrap b nel bucket appropriato
- Effettua il lookup della propria chiave (notificandosi)

Tradeoff tra informazione di routing e costo di ricerca



Fairness

FAIRNESS IN SISTEMI P2P

Fairness in sistemi P2P

Comportamenti opportunistici/egoistici degradano le prestazioni

- Incentivi/sanzioni e.g. dare priorità agli utenti che condividono maggiori risorse

Il dilemma dei prigionieri:

- Due sospetti sono stati arrestati e interrogati separatamente
- Non ci sono prove sufficienti per una condanna quindi viene proposto un accordo:
  - Se un sospettato accusa l'altro e questi rimane in silenzio il primo è libero il secondo 10 anni
  - Se entrambi si accusano sono condannati a 2 anni a testa
  - Se entrambi rimangono in silenzio sono condannati a 6 mesi
- Il comportamento razionale opportunistico (entrambi tradiscono) non è un ottimo di Pareto. L'ottimo corrisponde all'opposto.
- Versione iterativa con memoria: ottimo anche in presenza di opportunismo se razionale.

Tit-for-tat in BitTorrent

Pan-per-focaccia:

- Inizialmente ogni peer è disposto a cooperare con ogni altro peer
- Successivamente collabora solo con chi si è dimostrato collaborativo

In BitTorrent

- Cooperazione = consentire l'upload di file locali verso altri peer
- Tradimento = rifiuto di upload verso altri peer
- «chocking» = gli upload sono rifiutati ai peer che hanno tradito

Le interazioni p2p hanno una scala temporale superiore alle singole istanze di cooperazione

I tradimenti vengono perdonati col tempo o condonati periodicamente (cf giochi con memoria)

Fiducia e reputazione

Sistemi di incentivi per massimizzare la cooperazione dipendono dalla fiducia (reciproca e nel sistema).

Popolarità di un peer è variabile, instabile e non necessariamente immagine della qualità del servizio offerto.

Trust management in sistemi P2P:

- Nessun nodo ha una visione completa del sistema (spazio e tempo).
- Bisogna affidarsi ad altri peer per valutare l'affidabilità di un servizio offerto da un dato peer.
- Lo stesso meccanismo di gestione della reputazione può non essere affidabile (e.g. attacchi MIM).
- Assenza di metriche precise.
- Rischio di collusione.

Esercizi

Implementazioni DHT

- OpenChord, JDHT, OpenKad
- LibBT, BitDHT, maidsafe-dht
- Scalaris, Ringo

1. Chat distribuita
2. FileSystem rudimentale
   1. Flat
   2. Semantica a scelta
   3. Senza autenticazione/autorizzazione
3. Tuple space
4. Implementare un protocollo DHT (meglio in Erlang)