Cloud Computing

Cap 2. Multicast. Protocoale Gossip. Membership.

October 29, 2022

YARN[1]

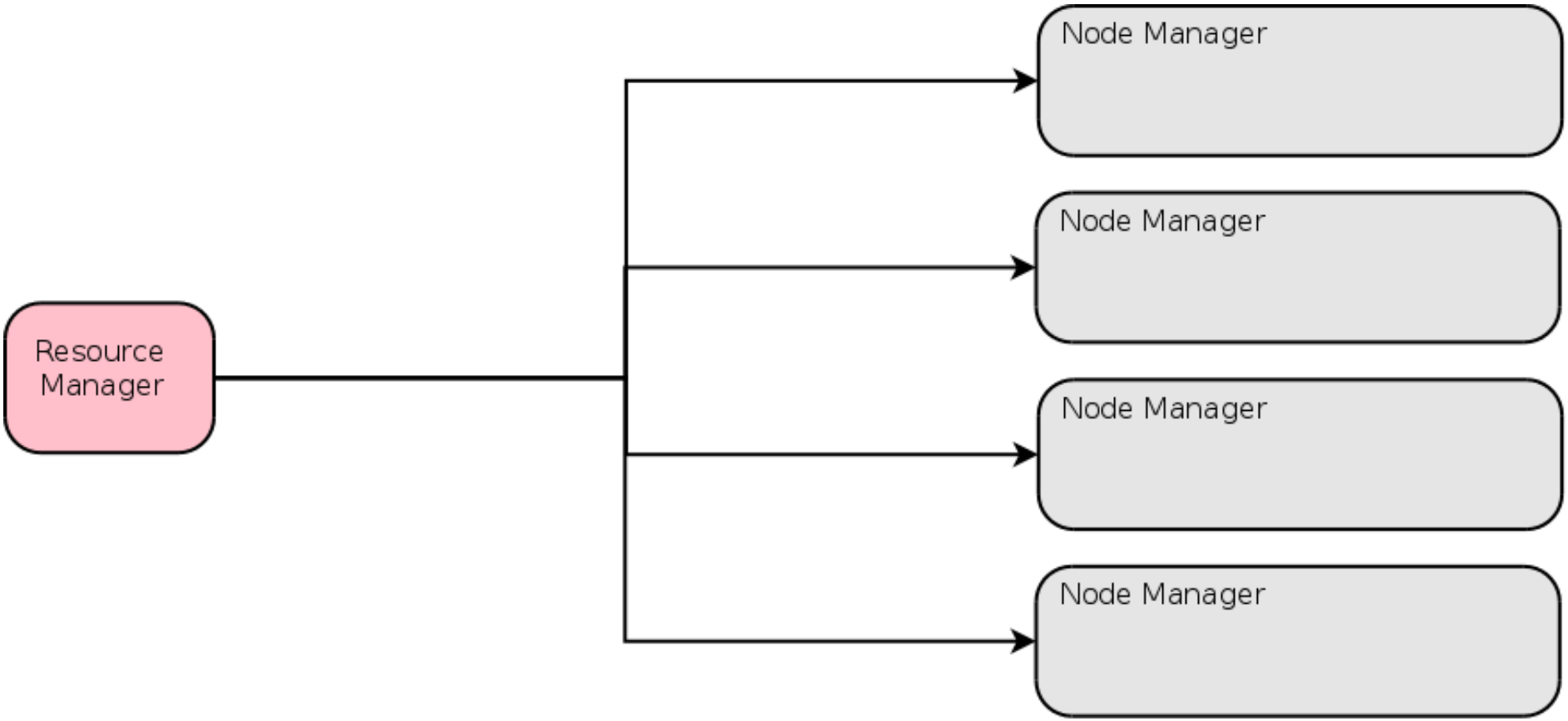
- folosit în Hadoop 2.x +
- fiecare server este văzut ca o colecție de containere
- fiecare container este un fel de thread ce dispune de putere de procesare + memorie
- componente:
  - Resource Manager (RM) - scheduler global
  - Node Managers (NM) - per node, daemon, funcții specifice
  - Application Master (AM) - alocat la cerere per node, negociază cu RM și NM; detectează job failure

[1] Yet Another Resource Negotiator

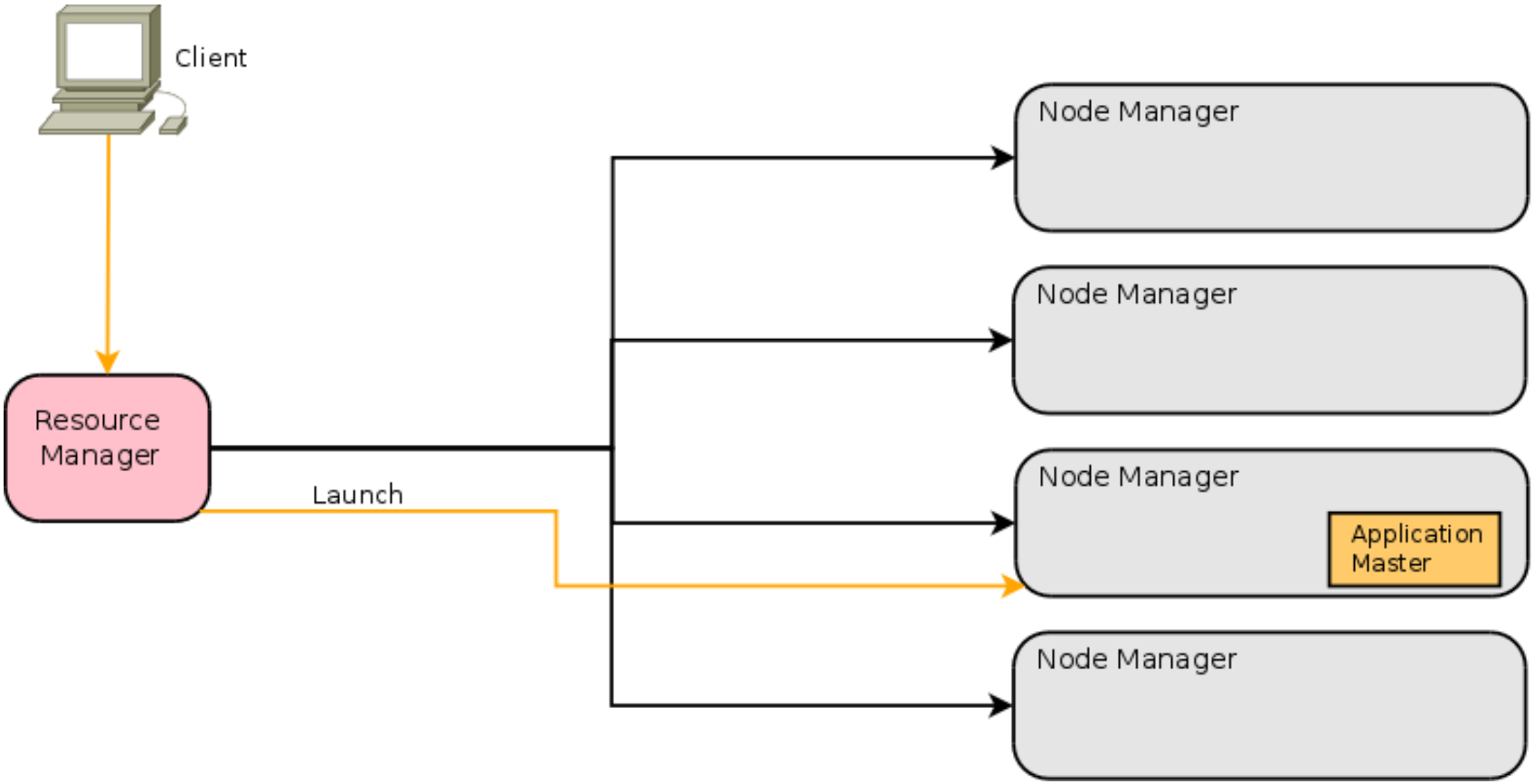
YARN daemons

- Resource Manager (RM)
  - rulează pe nodul master
  - scheduler global
  - arbitrează resursele sistemului între aplicațiile ready
- Node Manager (NM)
  - rulează pe nodurile slave
  - comunică cu RM
- Containers
  - create de RM la cerere
  - alocă o cantitate de CPU, memorie pe nod
  - aplicațiile rulează în unul sau m.m. containers
- Application Master
  - câte unul per aplicație
  - rulează în container
  - cere m.m. containere pt. a rula task-urile aplicației (mappers și reducers)

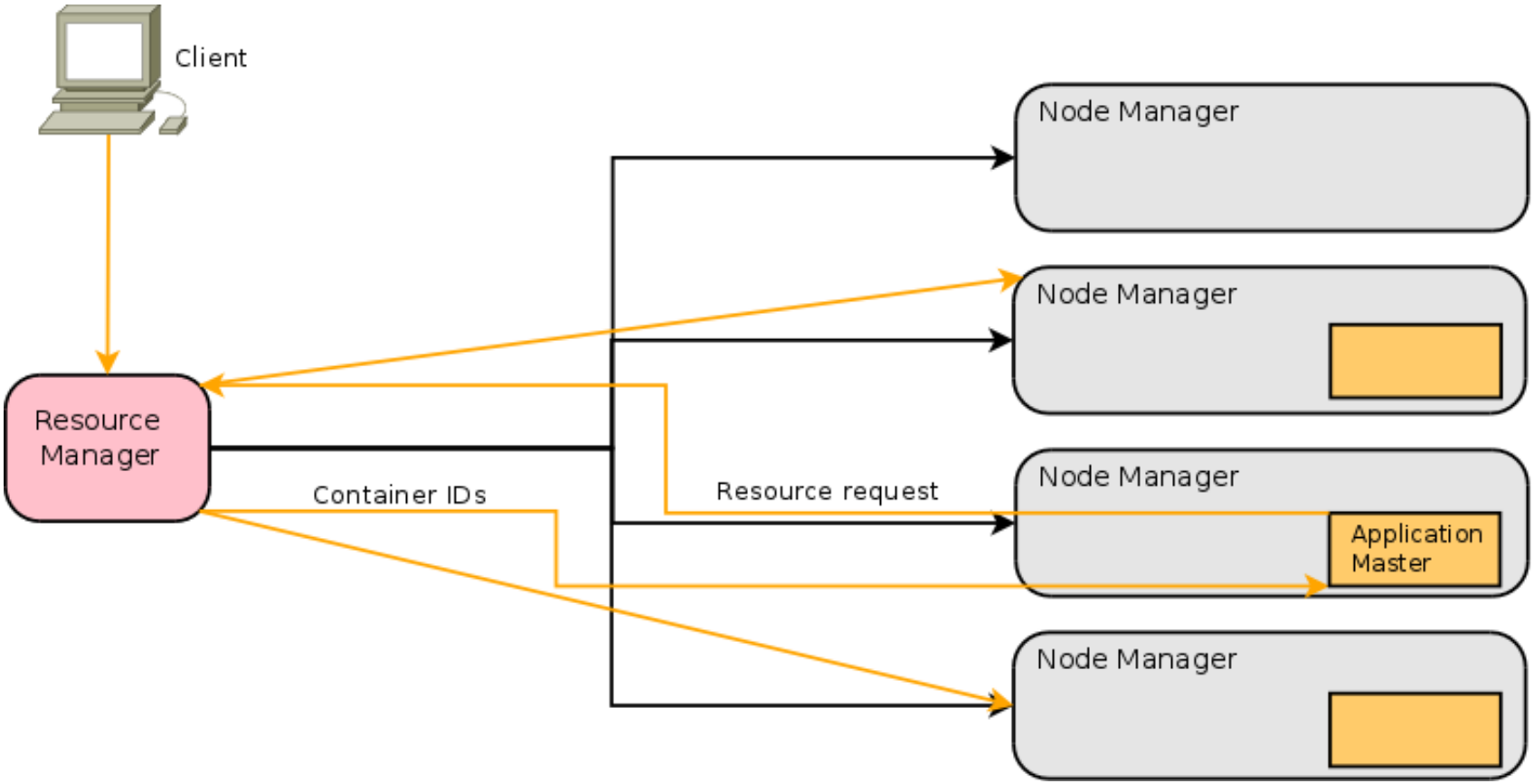
Job spawn


Resource Manager
Node Manager
Node Manager
Node Manager
Node Manager

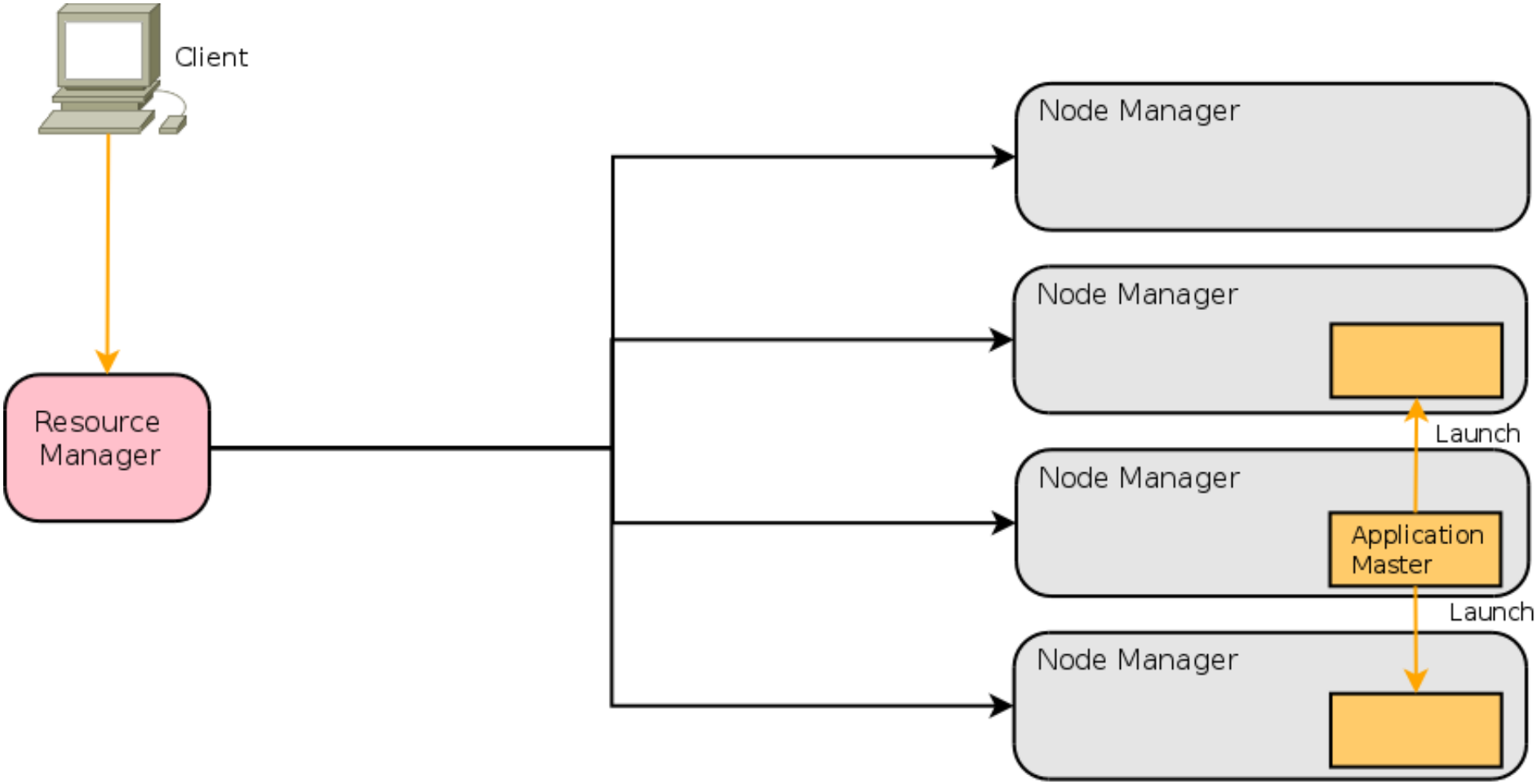
Job spawn


Client
Resource Manager
Launch
Node Manager
Node Manager
Node Manager
Application Master
Node Manager

Job spawn


Client
Resource Manager
Container IDs
Resource request
Node Manager
Node Manager
Node Manager
Application Master
Node Manager

Job spawn


Client
Resource Manager
Node Manager
Node Manager
Launch
Node Manager
Application Master
Launch
Node Manager

Fault tolerance

- task fail (container)
  - Application Master va re-rula task-urile care se termină cu excepții sau nu mai răspund, de 4 ori (default)
  - ulterior, aplicațiile cu task-uri failed sunt declarate failed
- Application Master fail
  - dacă AM nu mai trimite mesaje heartbeat, RM va re-rula aplicația de 2 ori (default)
- Node Manager fail
  - dacă NM nu mai trimite heartbeat la RM, este șters din lista de noduri active
  - task-urile de pe nod vor fi tratate ca failed de AM
  - dacă nodul AM pică, întreaga aplicație va fi tratată ca failed
- Resource Manager fail
  - nu se pot lansa aplicații / task-uri
  - poate fi configurat cu high-availability (al doilea RM în stand-by), în caz de cădere a celui primar se reface din checkpoint
- mesajele heartbeat - piggyback pt. requests pentru scăderea traficului

Slow servers

- cea mai înceată mașină încetinește întregul job
- cauze: disk, bandwidth, slow CPU, memorie ocupată
- se menține evidența progresului fiecărui task
- straggler task: task-ul cu cel mai lent progres
- execuție replicată a task-ului straggler pe mai multe noduri
- cea mai rapidă replică termină și marchează task-ul ca fiind complet (tehnică denumită speculative execution)

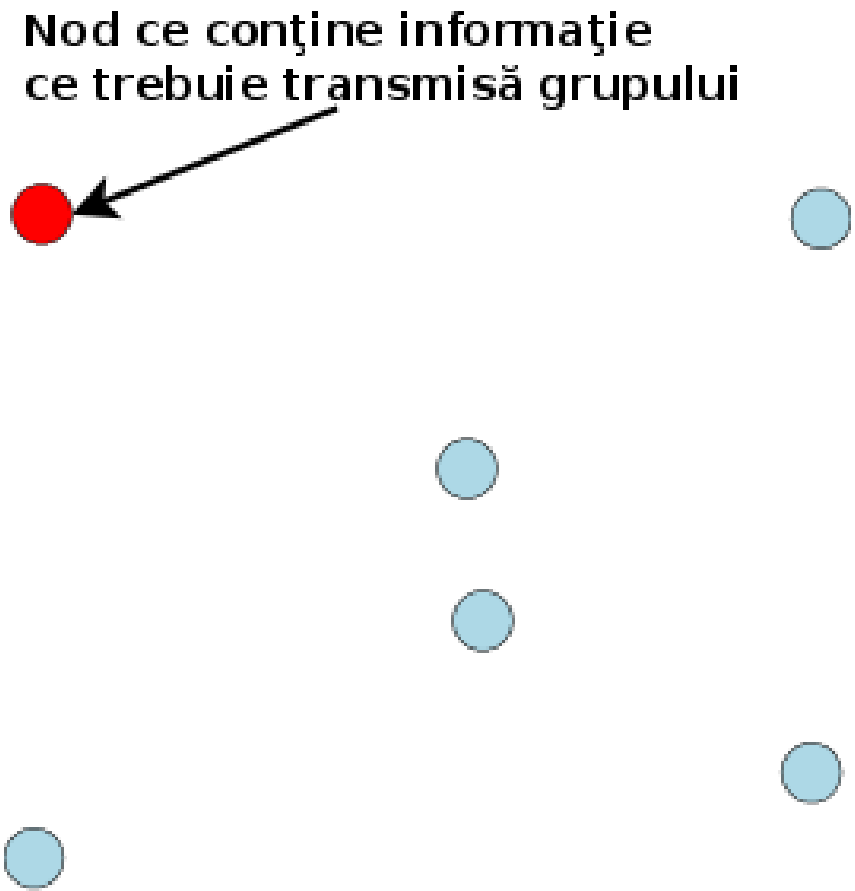
Locality

- cloud-ul are topologie ierarhică (rack switch vs. core switch)
- GFS / HDFS stochează 3 replici ale fiecărui chunk (shard redundancy)
- 32 / 64 MB fiecare chunk (mapper split implicit, dacă nu se fac mai multe fișiere)
- în rack-uri diferite (ex. chunk 1 într-un rack, chunk 2 într-un alt rack)
- la căderea unui task, se încearcă repornirea task-ului (în ordine)
  1. pe o mașină ce conține o replică a datelor, sau
  2. în același rack ca mașina ce conținea datele, sau
  3. oriunde

Problema multicast

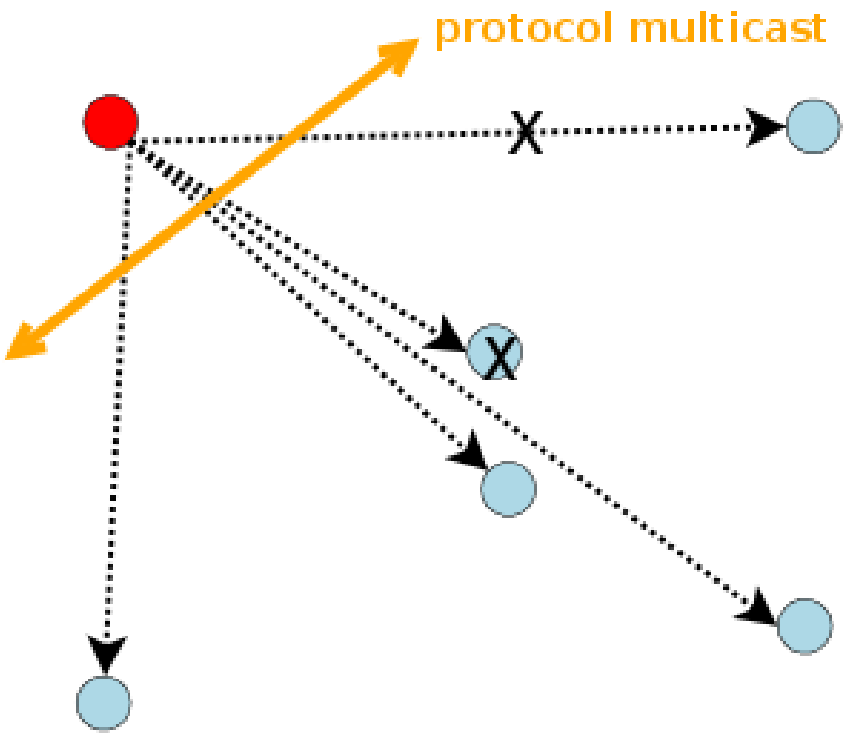
- clasă largă de probleme rezolvate de algoritmul Gossip
- multicast - una din probleme
- grup de noduri Internet
- fiecare nod poate trimite / recepționa mesaje
- problema: transmiterea eficientă a unui mesaj către toți membrii grupului

Problema multicast



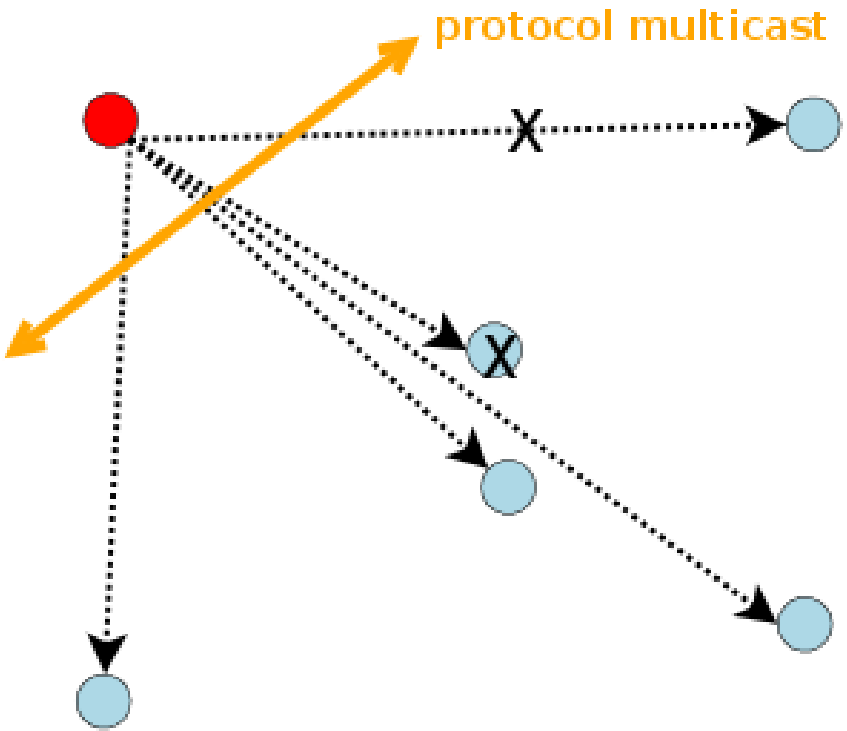
- grup distribuit de noduri
- host-uri conectate în Internet
- ex. distribuția în timp real a informațiilor bursei de valori
- mesajele multicast sunt simultane
- multicast (grup) vs. broadcast (all)

Cerințele algoritmului multicast



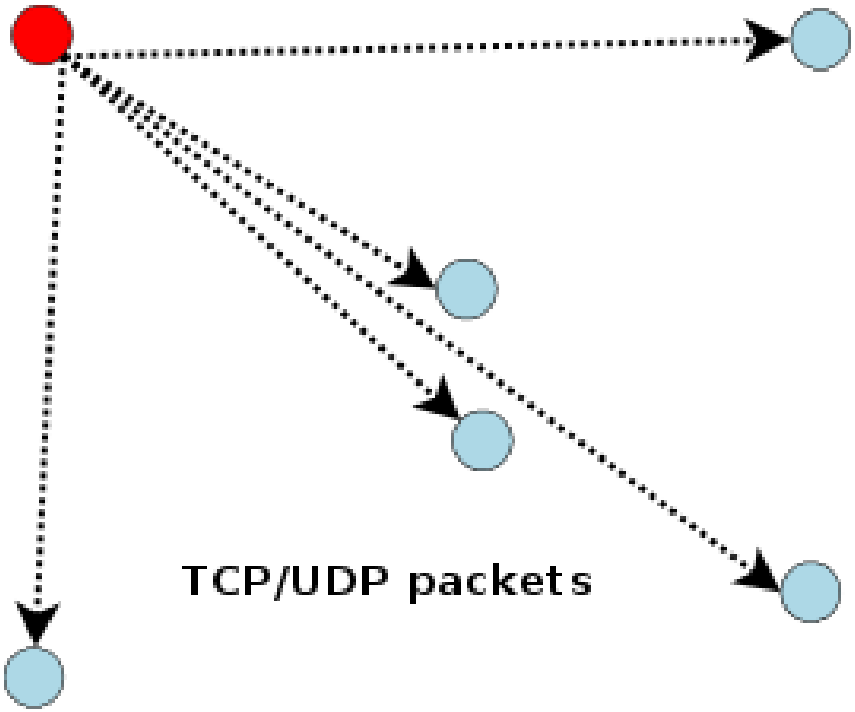
- fault tolerance și scalabilitate
- căderi de noduri, pierderi de pachete, întârzieri
- reliability, toți receiverii funcționali vor trebui să primească pachetele
- scalabilitate, overhead mic per node la creșterea numărului de noduri

Cerințele algoritmului multicast



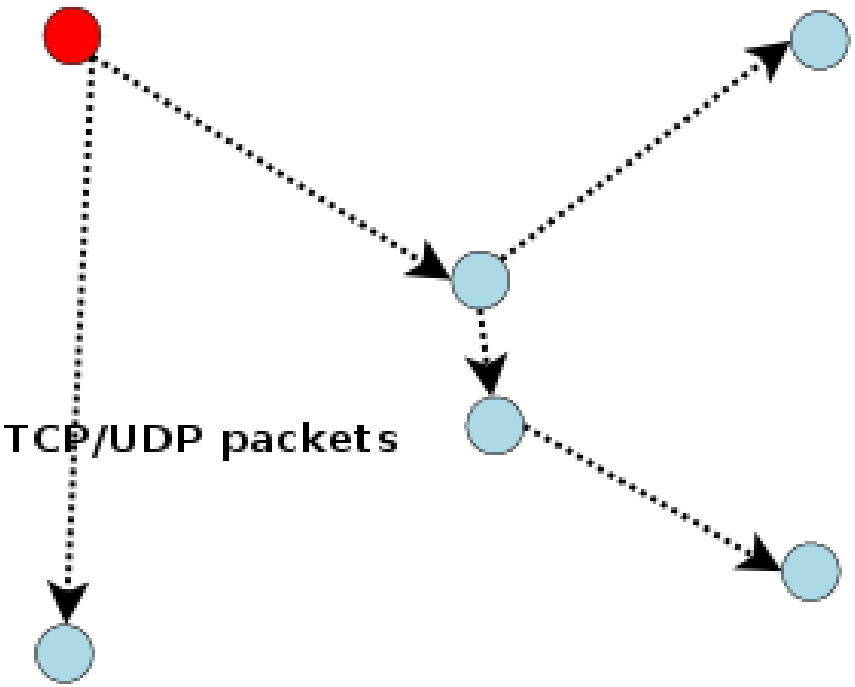
- protocol la nivel aplicație
- IP-Multicast la nivel rețea (uneori e folosit, sau neactivat la nivel de router)

Abordare centralizată



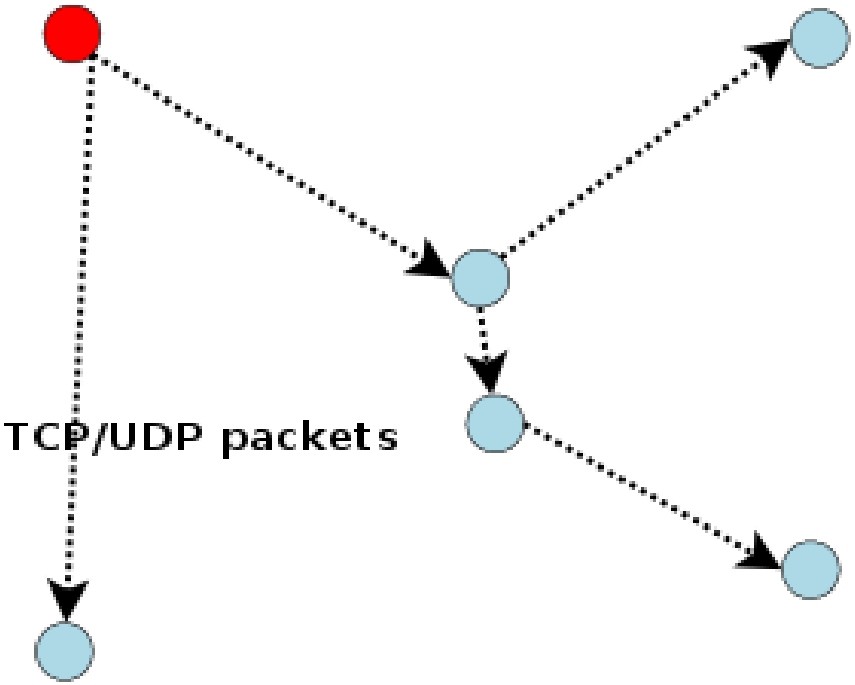
- sender + listă destinatari
- implementare simplă
- UDP (connection-less) vs. TCP (connection-oriented)
- fault tolerance: sender crashes?
- overhead la sender: mii de membri în grup, latența este în O(N)

Abordare tree-based



- spanning tree
- nivel rețea: IP-multicast
- nivel aplicație (papers):
  - SRM (Scalable Reliable Multicast - 1992)
  - RMTP (Real Time Messaging Protocol - Adobe Flash),
  - TRAM (Tree-based Reliable Multicast protocol - 1998),
  - TMTP (Tree-based Multicast Transport Protocol - 1995)

Abordare tree-based

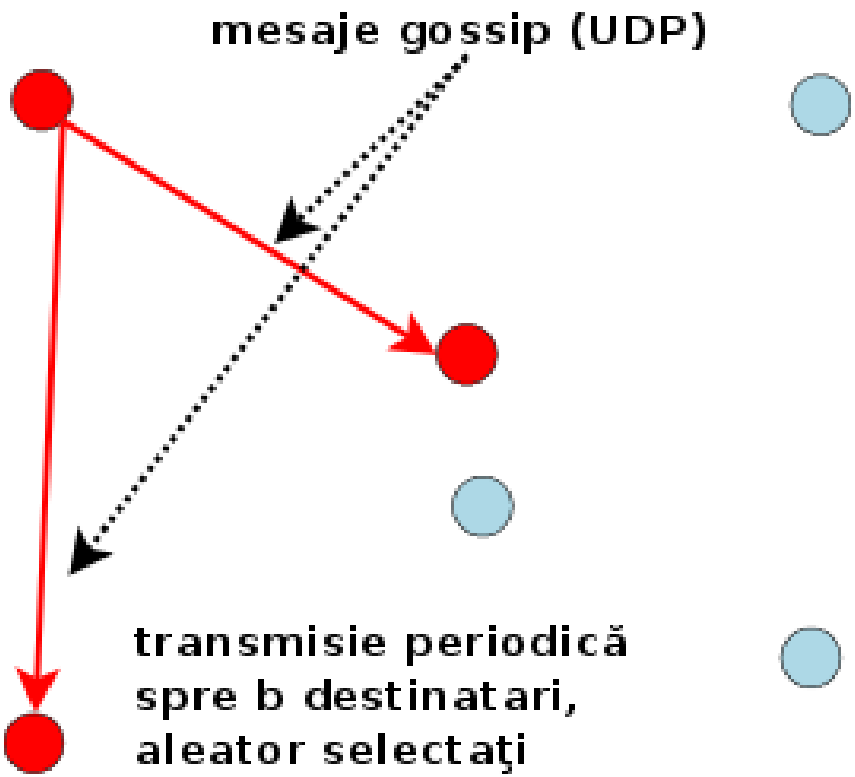


- arbore echilibrat, înălțime în O(log(n)) vs. centralized O(N)
- overhead individual per nod, constant
- failure: copiii nu vor mai primi mesaje pt. o vreme
- failure e un fapt cert

Protocoale multicast bazate pe arbori

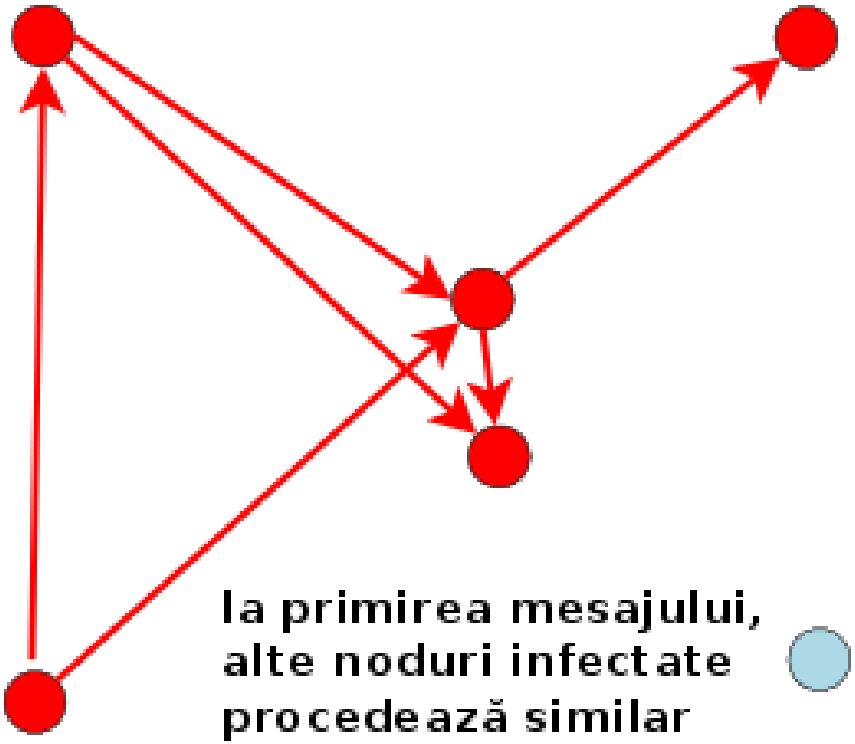
- asamblează nodurile din grup sub forma unui arbore cu număr fix de copiii per nod (spanning tree)
- mesajele multicast se trimit folosind acest arbore
- folosesc fie acknowledgements (ACKs) sau acknowledgement-urile negative (NAKs) pentru retransmitere
- SRM (Scalable Reliable Multicast)
  - folosește NAKs
  - NAK storms, folosește random delays, exponential backoff
- RMTM (Reliable Multicast Transport Protocol)
  - folosește ACKs
  - ACKs trimise doar unor receptori dedicați pentru a evita 'poluarea' cu mesaje
  - doar receptorii dedicați fac retransmisia packetelor lipsă
- pot totuși cauza overhead ACK / NAK în O(N)
- NAK nu se poatefolosi dacă suportul de IP-Multicast e activ (explozie de mesaje)

Protocolul Gossip



- pot exista mai mulţi senders în paralel
- tipic, b = 2 (gossip fan-out)
- posibil ca un destinatar să fie ales de mai multe ori
- sender: infectat, receiver: neinfectat (încă)

Noduri deja infectate



- destinatarul este selecatat aleator
- la final toate nodurile din grup for fi infectate
- mai multe mesaje decât necesar
- overhead nu foarte ridicat (vom vedea!)

Gossip = epidemic multicast

- inspirat din modalitatea de propagare a unei boli
- noduri infectate vs. noduri ne-infectate
- un nod infectat trimite mesaje identice către b receptori selectați aleator
- fiecare nod își rulează propriul algoritm independent de celelalte (nesincronizat)

Push vs. pull

- până acum, abordare push
  - după primirea unui mesaj multicast, nodul începe 'bârfa'
  - în cazul primirii mai multor mesaje, se trimit toate / un subset selectat random / cele mai importante
- pull Gossip:
  - un nod (infectat sau ne-infectat) periodic selectează aleator noduri și trimite solicitări de mesaje
  - acele noduri vor trimite ca răspuns mesajele noi
- variantă hibridă: push-pull

Analiza push Gossip

- overhead redus chiar și pentru grupuri mari
- împrăștie rapid un mesaj multicast (în puțini pași)
- fault-tolerance ridicat (d.p.d.v. al node failure-urilor sau mesajelor pierdute)

Analiza

- tratat de epidemiologie (Bailey, 1975)
- $n + 1$ indivizi în grup ce se amestecă omogen (ex. molecule amestecate într-un borcan)
- rata de contact între indivizii dintr-o pereche este β
- la orice moment, un individ este fie infectat fie neinfectat; sunt x indivizi neinfectați și y infectați
- $x_0 = n$, $y_0 = 1$; la orice moment, $x + y = n + 1$
- un nod odată infectat, rămâne infectat

Analiza

- proces stochastic, cu timp continuu; presupunem nodurile sincronizate $\frac{dx}{dt} = -\beta xy$
- variația e descrescătoare în timp, xy descrie numărul de contacte posibile, din care doar o fracție se soldează cu infectare
- soluția:

$$x = \frac{n(n+1)}{n + e^{\beta(n+1)t}}, y = \frac{(n+1)}{1 + ne^{-\beta(n+1)t}}$$

(de demonstrat!), unde

$$\lim_{t\to\infty} x = 0, \lim_{t\to\infty} y = n + 1$$

Analiza

- probabilitatea ca nodul respectiv să fie ales ca 'victimă' a infectării: $\beta = \frac{b}{n}$
- la timpul $t = c \cdot log(n)$, numărul de noduri infectate este: $y \approx (n+1) - \frac{1}{n^{cb-2}}$ (de demonstrat!)
- partea a doua este foarte aproape de 0

Analiza

- se aleg c, b mici și independente de n
- în $c \cdot log(n)$ runde (latență scăzută),
- toate în afară de $\frac{1}{n^{cb-2}}$ noduri primesc mesajul (reliability)
- un nod a transmis $c \cdot log(n)$ mesaje (overhead redus)

Discuție O(log(n))

- log(n) nu e o constantă
- practic, este o funcție cu creștere înceată
- baza 2:
  - log(1000) aprox. 10
  - log(1M) aprox. 20
  - log(1 miliard) aprox. 30
  - log(spațiul de adrese IPv4) = 32

Fault-tolerance pentru push Gossip

- packet loss
  - 50% pierdere de pachete; analiza reluată pentru b înlocuit cu $b/2$ (random, unul din două mesaje e pierdut)
  - pentru aceeași reliability (același y), de două ori mai multe runde (impact asupra lui c)
- node failure
  - 50% din noduri pică; se reia analiza cu n înlocuit cu $n/2$ și b înlocuit cu $b/2$ (jumătate din noduri și jumătate din mesaje)
  - numărul de mesaje crește de un număr de ori (constantă)

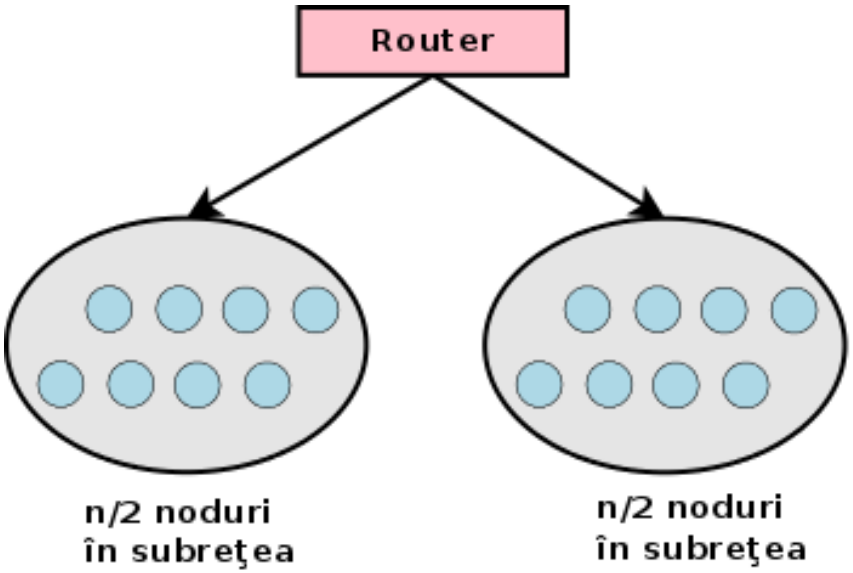
Fault-tolerance pentru push Gossip

- în condiții vitrege, e posibil ca epidemia să se 'stingă'?
- posibil, dar improbabil
  - 'stingerea' e cea mai probabilă la început (containment)
  - odată ce câteva noduri se infectează, cu probabilitate mare, epidemia va răbufni
  - analiza anterioară dă comportarea cea mai probabilă
- exemple:
  - zvonurile
  - boli infecțioase $\rightarrow$ epidemii
  - rata de propagare a virușilor / viermilor de Internet

Analiza pull Gossip

- în toate formele de gossip, e nevoie de O(log(n)) pași înainte ca aproximativ $n/2$ noduri să primească mesajul
- 'cauza' e dată de înălțimea arborelui de comunicație
- pull gossip va fi mai rapid decât push gossip:
  - fie p_i fracția nodurilor neinfectate la pasul i
  - probabilitatea ca nodul să fie neinfectat și la pasul $i+1$ e dat de probabilitatea ca să fi fost neinfectat,
  - și de probabilitatea de a 'scăpa', adică toate cele k contacte alese să fie și ele neinfectate:
    $p_{i+1} = p_i \cdot p_i^k = p_i^{k+1}$
  - super-exponential (v. funcția lui Ackermann și logaritmul stelat)
  - infectarea a $n/2$ noduri se face în log(log(N))

Analiza pull Gossip



- topologie ierarhică
- pentru selecție aleatorie a destinatarului, core router suportă o încărcare în O(n)
- soluția: în subrețeaua i, ce conține n_i noduri, se selectează un destinatar în subrețea cu probabilitatea $1 - 1/n_i$
- încărcarea core router-ului devine O(1) - după ce subrețeaua s-a saturat, fiecare din cei n_i trimite în afară cu probabilitatea $1/n_i$
- timpul de propagare în toată rețeaua rămâne în O(log(n))

Implementări Gossip

- proiectele Clearinghouse și Bayou: tranzacții email și database (Xerox, PODC 1987)
- refDBMS system (Usenix 1994)
- Bimodal Multicast (ACM TOCS, 1999)
- sensor networks, wireless networks (2002, 2005)
- AWS EC2 și S3 (2000)
- Cassandra key-value store pentru păstrarea listelor de membership
- NNTP (Network News Transpor Protocol, 1979)

Protocolul NNTP inter-server

- (client:) fiecare client uploadează / downloadează știri (posts) pe/de pe server
- (server upstream:) fiecare server trimite o listă de ID-uri de mesaje, cu mesaje pe care deja le are
- (server downstream:) se trimite drept răspuns o listă cu ID-uri de mesaje pe care server-ul downstream nu le are
- (server upstream:) ca răspuns, se trimit mesajele cu ID-urile solicitate
- (server downstream:) OK
- rolurile se inverseaza
- server-ul reține post-urile pentru o vreme, le transmite la cerere, apoi, după o vreme, le șterge
- informația persistă în rețea

Group membership - o provocare

- datacenters - manager-ul pretinde că nu există failures
- care e prima responsabilitate ca administrator?

Group membership - o provocare

- datacenters - manager-ul pretinde că nu există failures
- care e prima responsabilitate ca administrator?
- construcția unui mecanism de detectare a failure-urilor

Group membership - o provocare

- datacenters - manager-ul pretinde că nu există failures
- care e prima responsabilitate ca administrator?
- construcția unui mecanism de detectare a failure-urilor
- ce se poate întâmpla dacă nu se ia această măsură?

Failures - regulă, nu excepție

- rata de failure a unei mașini este de 1 / 10 ani (120 luni), în medie
- pentru 120 de servere în DC, MTTF (Mean Time To Failure) devine 1 mașină / lună
- la 12000 servere?

Failures - regulă, nu excepție

- rata de failure a unei mașini este de 1 / 10 ani (120 luni), în medie
- pentru 120 de servere în DC, MTTF (Mean Time To Failure) devine 1 mașină / lună
- la 12000 servere?
- MTTF = o mașină / 7.2 ore ! aproape continuu
- data loss (server down), software-ul nu a detectat acest lucru
- inconsistența datelor (pagube provocate clienților, nu numai downtime !)

Sistem de detecție a failure-urilor

- program distribuit de detecție a căderilor de noduri ce detectează automat failures și raportează / ia măsuri de corecție
- sisteme bazate pe grupuri
  - clouds / DCs
  - servere replicate (sisteme peer-to-peer de replicare a fișierelor)
  - baze de date distribuite
- crash-stop: nodul pică fără notificare, fără salvarea stării

Seviciul de tip group membership

- folosit de application queries: gossip, DHT (Distributed Hash Tables)
- protocol de păstrat și distribuit apartenența la membership list
- comunicație unreliable
- nodurile pot cădea pe neașteptate

Tipuri de membership

- strongly consistent membership: lista completă în orice moment (e.g. virtual synchrony, bazat pe state machines)
- **weakly consistent**: listă aproape completă (e.g. de tip gossip, SWIM)
- partial-random list: e.g. SCAMP, T-MAN, Cyclon

- (i) componenta **fault detection** (detectarea nodurilor nefuncționale = membership list)
- (ii) componenta de **diseminare** (propagarea listei de membership tuturor nodurilor)

- e.g. un nod pică; unul din nodurile sănătoase detectează acest lucru (i) și apoi folosește serviciile (ii) pentru a informa toți membrii grupului

Failure detectors

- nu se poate preveni crash-ul; ne vom pregăti pentru el
- apariție frecventă
- mai degrabă regula decât excepția
- frecvența crește liniar cu mărimea datacenter-ului

Proprietăți ale failure detectors

- **completness**: ca fiecare failure să fie detectată
- **accuracy**: să nu existe false positives, adică semnalări de false căderi
- viteza: timpul până la prima detecție a căderii nodului
- scalabilitate: încărcare echilibrată a nodurilor și încărcarea comunicării rețea

Proprietăți ale failure detectors

- **completness**
- **accuracy**
  - nu pot fi satisfăcute simultan
- viteza
- scalabilitate

Proprietăți ale failure detectors

- **completness**
  - garantată
- **accuracy**
  - parțială / garanție probabilă
- viteza
- scalabilitate

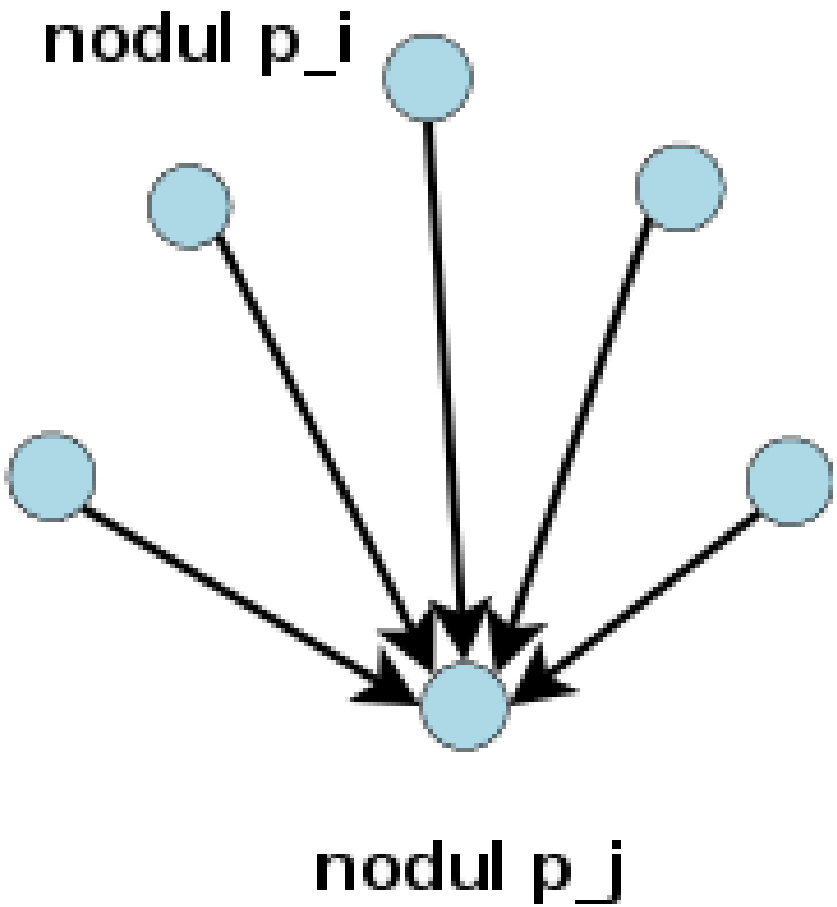
Proprietăți ale failure detectors

- **completness**
  - garantată
- **accuracy**
  - parțială / garanție probabilă
- viteza
  - timpul până la prima detecție să fie cât mai mic posibil
- scalabilitate

Proprietăți ale failure detectors

- **completness**
  - garantată
- **accuracy**
  - parțială / garanție probabilă
- viteza
  - timpul până la prima detecție să fie cât mai mic posibil
- scalabilitate
  - fără single-point failures / bottlenecks
- mai multe procese (noduri) pot pica simultan !

Heartbeating centralizat

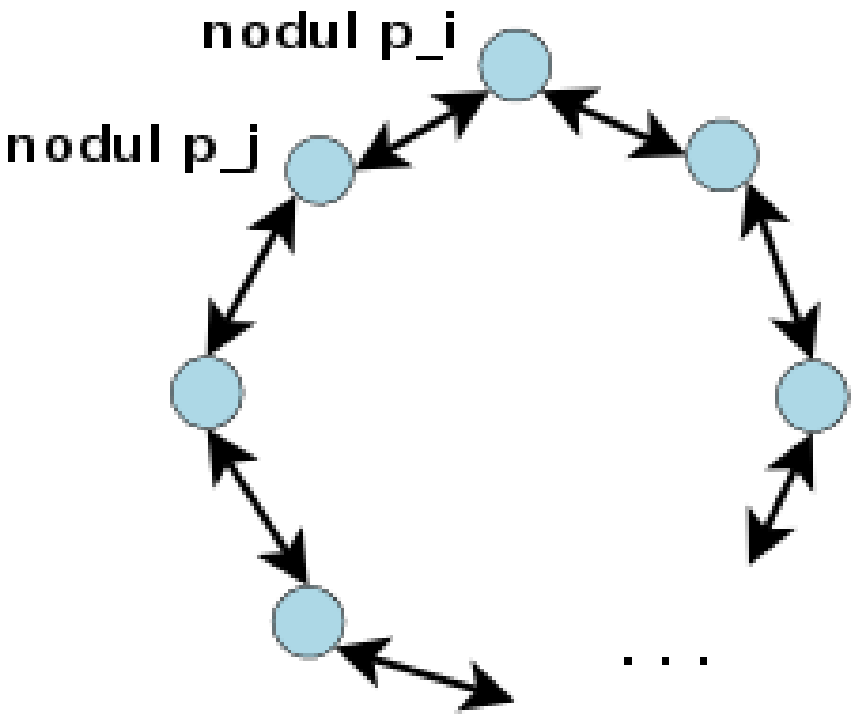


- p_i pică
- heartbeat-uri trimise periodic către p_j
- sequence number incrementat periodic la fiecare heartbeat
- p_j ține evidenta timpului ultimului heartbeat trimis de p_i
- la expirarea timpului, p_i este marcat ca failed

* dacă p_j pică, nu există garanții

* p_j supraîncărcat cu mesaje

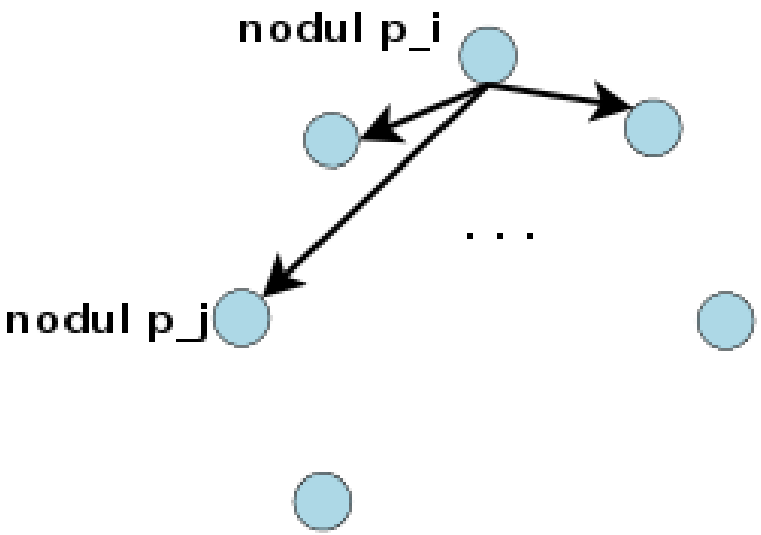
Heartbeating de tip ring



- heartbeats trimise spre vecinii imediați
- aceleași sequence numbers

* dacă vecinii lui p_i cad, o cădere între timp a lui p_i , trece nedetectată

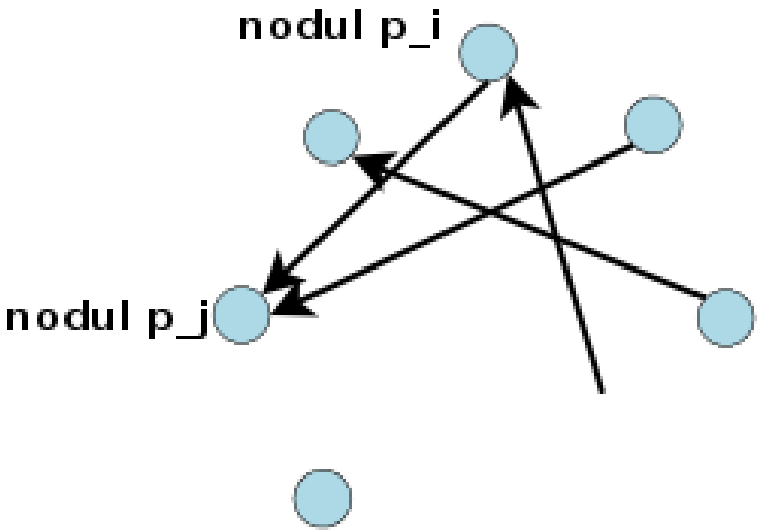
Heartbeating de tip all-to-all



- fiecare p_i trimite heartbeats către toate celelalte noduri din sistem
- fiecare proces va primi numere de secvență de la fiecare alt proces
- la expirare, procesul va detecta failure-ul

* încărcare mare, egal distribuită
* protocolul este complet; dacă cel puțin un nod rămâne viabil, acela va detecta failures
* un nod mai lent - celelalte noduri vor fi marcate ca failed - high FP rate

Heartbeating de tip gossip



- variantă de all-to-all heartbeating
- fiecare nod trimite heartbeats către un număr constant de vecini aleși random
- heartbeat-urile sunt numere de secvență incrementate local
- nodurile așteaptă heartbeats și marchează failure la expirare

* păstrează avantajele all-to-all heartbeating, dar cu o încărcare mai redusă

Failure detection cu gossip (1)

current time at node (2)	70		
	node	heartbeat counter	local time
membership list received from node (1)	1	**10120**	**66**
	2	10103	62
	3	**10098**	**63**
	4	10111	65
local membership list of node (2)	1	**10118**	**64**
	2	10110	64
	3	**10090**	**58**
	4	10111	65
computed new membership list of node (2)	1	**10120**	**70**
	2	10110	64
	3	**10098**	**70**
	4	10111	65

Failure detection cu gossip (2)

- dacă nu s-a primit heartbeat în ultimele t_{fail} secunde, nodul este considerat failed
- după $t_{cleanup}$ secunde, nodul este șters din listă
- de ce e nevoie de doi timpi de expirare?

Failure detection cu gossip - ștergerea nodului expirat

current time at node (2)	75		
t_{fail} at node (2)	24		
	node	heartbeat counter	local time
membership list received from node (1)	1	10120	66
	2	10103	62
	3	**10098**	**55**
	4	10111	65
local membership list of node (2)	1	10118	64
	2	10110	64
	3	**10098**	**50**
	4	10111	65
computed new membership list of node (2)	1	10120	75
	2	10110	64
	-	-	-
	4	10111	65

Failure detection cu gossip - falsa reînnoire

current time at node (2)	80		
t_{fail} at node (2)	24		
	node	heartbeat counter	local time
membership list received from node (1)	1	10120	66
	2	10103	62
	3	**10098**	**55**
	4	10111	65
local membership list of node (2)	1	10120	75
	2	10110	64
	-	-	-
	4	10111	65
computed new membership list of node (2)	1	10120	75
	2	10110	64
	3	**10098**	**80**
	4	10111	65

Failure detection cu gossip - analiza

- un heartbeat are nevoie de O(log(n)) pentru propagare
- n heartbeats au nevoie de:
  - O(log(n)) pentru propagare, dacă lățimea de bandă disponibilă per nod este în O(n)
  - O(n log(n)) pentru propagare, dacă lățimea de bandă disponibilă per nod este doar în O(1)
- reducerea $t_{gossip} \rightarrow$ lățime de bandă mai mare, timp mai scurt de propagare (detecție)
- dacă există constrângerea ca lățimea de bandă să fie mică, t_{gossip} va trebui să crească
- dacă t_{fail}, $t_{cleanup}$ cresc, se reduce FP rate ($p_{mistake}$), dar timpul de detecție crește

Soluția ecuației diferențiale

$x_0 = n \qquad y_0 = 1 \qquad x + y = n + 1 \qquad \frac{dx}{dt} = -\beta xy$

fie $n + 1 = m$ și $y = m - x$; avem: $\frac{dx}{dt} = -\beta x(m - x)$,

de unde $\frac{dx}{x(m-x)} = -\beta dt$, dar:

$$\frac{1}{x(m-x)} = \frac{A}{x} + \frac{B}{m-x} = \frac{1}{mx} + \frac{1}{m(m-x)}$$

, ceea ce duce la:

$$\int \left[\frac{1}{mx} + \frac{1}{m(m-x)}\right] dx = -\beta \int dt$$

$$\frac{ln(mx)}{m} + \frac{ln[m(m-x)]}{-m} = -\beta t + c$$

$$\frac{1}{m} ln \frac{mx}{m(m-x)} = -\beta t + c \iff ln \frac{x}{m-x} = -\beta mt + mc$$

$$\frac{x}{m-x} = Ae^{-\beta mt} \iff x = (m-x)Ae^{-\beta mt}$$

Soluția ecuației diferențiale

$$(1 + Ae^{-\beta mt})x = Ame^{-\beta mt} \implies x = \frac{Ame^{-\beta mt}}{1 + Ae^{-\beta mt}}$$

$$n = x(t = 0) = \frac{Am}{1 + A} \qquad n(1 + A) = mA$$

$n = (m - n)A = A \implies A = n$, deci:

$$x = \frac{n(n+1)e^{-\beta mt}}{1 + ne^{-\beta mt}} = \frac{n(n+1)}{n + e^{\beta(n+1)t}} \tag{1}$$

$$y = \frac{(n+1)(1 + ne^{-\beta mt} - ne^{-\beta mt})}{1 + ne^{-\beta mt}} = \frac{n+1}{1 + ne^{-\beta(n+1)t}} \tag{2}$$

Analiza numărului de noduri infectate

$$fie\ \beta = \frac{b}{n} \qquad t = clog(n) \qquad n = e^{log(n)}$$

$$y = \frac{n+1}{1 + ne^{-\frac{b}{n}(n+1)clog(n)}} \approx \frac{n+1}{1 + \frac{n}{(e^{log(n)})^{bc}}} = \frac{n+1}{1 + \frac{1}{n^{bc}n^{-1}}}$$

$$y = \frac{n+1}{1 + \frac{1}{n^{cb-1}}} = (n+1)\frac{1 - \frac{1}{n^{cb-1}}}{1 - \left(\frac{1}{n^{cb-1}}\right)^2} \approx (n+1)\left(1 - \frac{1}{n^{cb-1}}\right)$$

$$y \approx (n+1) - \frac{1}{n^{cb-1} \cdot n^{-1}} = (n+1) - \frac{1}{n^{cb-2}}$$