Sistemas Distribuidos

- Un sistema distribuido es un conjunto de maquinas que coordinan sus acciones y que aparecen al usuario como un sistema unico y coherente.
- emplea software sobre la red para lograr su cometido: middleware.

Componentes autonomos: computadores, impresoras, etc.
Paso de mensajes: medio unico de interaccion es el mensaje que se transmite por un canal de comunicacion (red de computadoras).
Usuarios: personas o programas que interactuan con un sistema unico.

- porque distribuir:
  - → confiabilidad: tolerancia a fallas / disponibilidad.
  - → escalable: performance

- Para aumentar la performance, incremento las replicas
- La consistencia puede ser a nivel de usuario o de datos.
- Un sistema debe ser ESCALABLE, es decir, que estos se puedan agrandar o achicarse de ser necesario.
  - → horizontal: colocar mas equipos
  - → vertical: es aumentar la RAM.

- Las replicas ayudan a la performance y a la tolerancia a fallos.
- Tecnicas de escalabilidad:
  - → particionar y distribuir.
  - → generar replicas.
  - → cache (memoria pequeña y de acceso rapido.) / limitado.
    - agarrar lo relevante y ponerlo en memoria cache.

- Objetivos de diseño:

1. Poner recursos a disposicion:
   - → facilitar a los usuarios y aplicaciones acceder recursos remotos de manera eficiente.
2. Hacer transparente la distribucion:
   - → esconder el hecho que procesos y recursos estan fisicamente distribuidos en multiples computadores.
   - → transparencia:

a) acceso: o/s tienen diferentes nomenclatura para los archivos y como se manipulan.
b) ubicacion: donde se encuentra el recurso.
c) migracion: recurso puede ser movido y no acepta su localizacion
d) Re-localizacion: recursos en uso de movido a otra ubicacion.

e) Replicacion: esconde el hecho que existan multiples copias.
f) Concurrencia: dos usuarios no notan que ambos estan accediendo a un recurso.
g) Falla: usuario no nota que el sistema falla, ni percibe que se recupera.

3. Ser abierto:
→ Sistema distribuido abierto es aquel que ofrece servicios bajo reglas estandar (sintaxis, semantica).
• Interoperabilidad: permite que procesos se comuniquen y generan implementaciones diferentes.
• Portabilidad: extensiones pueden ser usados en otros sistemas.
• Facil extension: agregar mas componentes.

4. Ser escalable:
→ Habilidad del sistema para expandirse para satisfacer las necesidades de su negocio. Un sistema puede escalar agregando HW adicional o actualizando HW existente sin cambiar gran parte de la aplicacion.
→ escalabilidad horizontal: aumentar la cantidad de los recursos. ej: mas PCs
→ escalabilidad vertical: aumentar las capacidades de mi recurso. ej: mas memoria
→ 3 tecnicas para escalar:
• Particion: dividir un componente y distribuirlo.
• Replicar: hacer copias de los recursos en distintos lugares.
• Cache: subconjunto de recursos de acceso facil y rapido.

• Particionamiento: termino genenco que simplemente significa dividir sus entidades logicas en diferentes entidades fisicas para rendimiento, disponibilidad o algun otro proposito.
→ particionamiento vertical: me centro en un concepto.
→ " horizontal: me centro en todos.

• Replicación: Compartir información para garantizar coherencia entre recursos redundantes, como SW y HW para mejorar la confiabilidad, la tolerancia a fallas o la accesibilidad.
→ Copia de un recurso que reside en un lugar diferente.

• Cache: espacio reducido de almacenamiento para la información de manera temporal y de acceso rapido.
→ dependiendo de lo que se guarde, influye en el rendimiento.
→ politicas de remocion: (LRU, FIFO, Random, MRU, etc).
→ politicas de ingreso: mecanismo de control de acceso, ley de potencias, analiza datos en frecuencia, etc.

→ tiempo de vida (Time to live): depende de la aplicación y el tiempo para ver cuanto tiempo mantengo una entrada a caché.

Comunicación:
→ medio de comunicación es el paso de mensajes por medio de la red.
→ red saturada, comunicación fiable no y hace difícil el desarrollo de sistemas distribuidos.
→ fundamentos de comunicación:
* La red, tiene varios dispositivos HW, varios componentes SW, medio físico.
* performance:
- Latencia: tiempo de transmitir un mensaje vacío (T).
- tasa de transferencia: velocidad de transmisión de datos (tT).
- tiempo para transmitir un mensaje: M(T.M) = T + b/tT.
- jitter: variación de tiempo que toma entregar una serie de mensajes.

→ LAN: Local Area Network: Alta velocidad.
→ MAN: Metropolitan Area Network: Alta velocidad.
→ WAN: Wide Area Network: Velocidad menor.
→ Wireless Networks: utiles para dispositivos moviles.
→ Internetworks: conexión de varios subsistemas de redes, util para el desarrollo de SD extensibles.

Para comunicarse existen 2 protocolos:
→ orientados a conexión.
→ no orientados a conexión.

Modelo OSI: 7 capas
- capa física: transmitir 1 y 0.
- capa enlace de datos: detectar, corregir
- capa de red: protocolo IP, envio paquetes y ruteo basado en IP address.
- capa de transporte: red no fiable, se encarga de dar conexion end-to-end particiona el mensaje y lo envia.
• middleware: SW informatico que proporciona servicios a aplicaciones de SW mas alla de los disponibles en el S.O.
Para comunicarse en un S.D existe entre interprocces comunication, indirect comunications, remote invocations.
1. Interprocess Comunications: comunicacion de bajo nivel entre procesos distribuidos, se utilizan primitivas, accesos directo a API entregadas entre protocolos internet.
2. Indirect Comunications: emisor y receptor no necesitan existir al mismo tiempo existe una tercera parte que interviene, e y r no estan al tanto de lo que

3. Remote Invocation: técnicas basadas en intercambios de dos sentidos (emisor-receptor), request/reply patron compuesto a un servicio basado en mensajes passing bajo un modelo c/s.

→ Remote procedure call (RPC): protocolo usado por un programa para solicitar un servicio de un programa ubicado en otra computadora en una red sin tener que comprender los detalles de la red.

- permite el llamado a procedimientos alojados en otra maquina.
- proceso A invoca un procedimiento en maquina B, define parametros y entrega resultados.

• Propiedades:

→ facil de entender, distribucion transparente, garantia ante fallas,

• Fallas:

→ congestion de red o servidor, caidas de clientes o error de programador.

• desventajas:

→ usa solo c/s, cliente bloqueado debe esperar, no es orientado a objetos, no es full transparente, aumento de latencia overhead.

• Problemas:

→ diferentes espacios de direcciones, caida de maquinas.

• consideraciones:

→ paso de parametros por valor o por referencia, ya que en ciertos C enteros y otros caracteres se pasan por valor, mientras que arreglos por referencia.

• Stub Cliente/Servidor:

Objetivo RPC generar llamadas a un procedimiento mas parecido a una local, quien llama no nota que se ejecuta de manera remota, librena con llamadas especificas.

• stub servidor:

recibe y transporta llamadas en locales; se bloquea al recibir una invocacion y espera el mensaje bloqueado; llama al proceso local con los parametros, los ejecuta y obtiene resultados.

• stub cliente:

recibe el mensaje y desempaqueta el resultado; se realiza una llamada a un procedimiento de manera tradicional y no hay invocacion a send y receive.

• marshalling: transformacion de los datos a un formato adecuado para transmision de un mensaje.

→ paso a parametros por valor: representar los datos (ASCII).

→ paso de parametros por referencia: servidor recibe el arreglo y lo referencia en su espacio local permitiendo ver los cambios, este se transmite por la red hacia el cliente.

PROARTE

Sincronización:

→ se necesita sincronización, cuando varios procesos deben decidir el orden de eventos o accesos a recursos compartidos.

- tiempos, relojes y sincronización de relojes:

Relojes físicos y lógicos:

→ **relojes físicos:**

- cristal + 2 registros (almacenador y contador).
- cada oscilación decrementa el contador, al llegar a cero se genera una interrupción y reinicia el valor del registro almacenador, cada interrupción es llamado tic de reloj.
- cuando se inicia el computador se solicita la fecha y hora que se traduce a un número de tics.
- avanzar la hora es posible, retrasarla no.

→ **Algoritmo de Cristian:** (UTC servidores sincronizados).

- Cada maquina pregunta el timeserver cada d/2p segundos la hora, el servidor responde con su hora actual y cuando recibe el cliente debe actualizar su reloj. Si la hora actual es posterior a UCT se ralentiza el reloj, si la hora actual es anterior a lo recibido, lo adelanta al UCT o lo acelera progresivamente.

→ **Algoritmo de Berkeley:**

- Pensado en ambientes sin acceso a UTC, mantener sincronizadas las maquinas a una misma hora, servidor no es pasivo (pregunta la hora periodicamente).
- Se debe considerar el promedio tolerante a fallas, solo utiliza un subconjunto de los tiempos recibidos para calcular el promedio.

→ **NTP: Network Time Protocol**

- Usa técnicas para filtrar los tiempos de calidad.
- Utiliza el algoritmo de Marzullo, el cual se utiliza para seleccionar fuentes de tiempo, relojes, para sincronizar tiempo entre varios relojes.
- servidores con estructura jerarquica y los niveles se conocen como stratum.
- 3 metodos de sincronización:
  1. Client-server: servidor chequea 1 o mas servidores para sincronizar.
  2. Symmetric: estratos bajo la arquitectura.
  3. Bradcast/multicast: se usa cuando se requiere poca precisión.

PROARTE®

conceptos:

- delay: tiempo de ida y vuelta desde que el cliente inicia hasta que recibe la respuesta.
- offset: diferencia de tiempo
- filter: error maximo del reloj del peer en relacion al reloj local y la ruta de red que los separa.
- poll: indica el intervalo de contacto.
- when: indica el numero transcurridos desde el ultimo contacto.
- type: indica el tipo de sincronizacion.
- Reach: registro de cambio que se utiliza para determinar el estado de accesibilidad del peer.

→ Relojes Logicos: (sincronizacion tiempo real).

- no es necesario sincronizar procesos que no interactuan, no es necesario que todos los procesos concuerden con el tiempo, sino que deben concordar con el orden en que ocurren los eventos.
- relojes logicos de lamport:

a ~ b ocurre antes que b

1) Si a y b son eventos en el mismo proceso, y a ocurre antes que b, entonces a ~ b es cierto.

La relacion entre antes (~) es transitiva.

a ~ b, b ~ c → entonces a ~ c

2) Si x e y ocurren en procesos diferentes que no intercambian mensajes; x ~ y no es cierto; y ~ x tampoco, son concurrentes, es decir, no se puede saber el orden de como ocurrieron.

tiempo C(a) manera de medir el tiempo, estos deben cumplir a ~ b es decir, c(a) < c(b).

El reloj C del proceso se incrementa en una unidad antes que se produzca el evento Cp = Cp + Δ, cuando un proceso envia un mensaje, se le adjunta el valor actual de t = Cp.

- multicast entre procesos:

→ ubica mensajes en su cola de acuerdo al timestamp, receptor envia un ACK, luego en el sgte lamport, si el timestamp del mensaje recibido tiene que ser menor al ACK.

→ misma copia de cola.

Reloj vectorial:

- CV(a) < CV(b) entonces a ~ b

Reloj logico

- a ~ b, entonces C(a) < c(b), pero c(a) < c(b) no implica que

Arquitecturas:

→ tipo está asociado a cual es la finalidad
→ 8 tipos:
- Computación distribuida
- Sistema de información
- sistemas distribuidos embebidos

Grid vs Cloud
- cloud público (AWS, Azure, GoogleCloud, etc).

→ no hay información / garantías en ubicación, multi-tenancy, rendimiento real.
- cloud privado:

→ permite monitoreo y toma de mediciones, no hay control sobre la configuración de la infraestructura.
- Grid:

→ se pueden modificar todos los niveles (VM, OS, Network).

Peer-to-Peer

→ comunicación directa, sistema abierto, completamente descentralizado.
→ 3 tipos: estructurada, no estructurada, híbrida.
→ ejecución de programas que no se pueden ejecutar de manera centralizada, restricciones de computo.
→ compartición de recursos de cálculo
→ descomposición de la aplicación en micro tareas.

Sistemas de información:

→ sistemas principalmente para el manejo e integración de funciones del negocio.
→ EIS: cualquier sistema de información que mejora las funciones de un proceso de negocio.
→ procesadores de transacciones: primitivas especiales, atómicos, consistentes, aislados, durables (ACID).
→ componentes o servicios SOA.

Sistemas Pervasivos:

→ Incluyen la componente de movilidad.
→ no hay control centralizado, características especiales: energía, procesamiento, almacenamiento, etc.

Arquitecturas Distribuidas

→ arquitectura del sistema:

Según colocamos los componentes del SW:.

- arquitecturas centralizadas: Cliente/Servidor
- arquitecturas descentralizadas: Peer-to-Peer
- arquitecturas hibridas: Edge

- **Cliente/Servidor**

→ 2 subsistemas o procesos

→ clientes: proceso que solicita un servicio.

→ servidores: proceso que implementa un servicio en especifico.

→ la app es modelada como un conjunto de servicios provedidos por los servidores y un conjunto de clientes que usan los servicios.

→ escalamiento vertical, modelo seguro, cuellos de botella posible.

→ mucha responsabilidad en el cliente

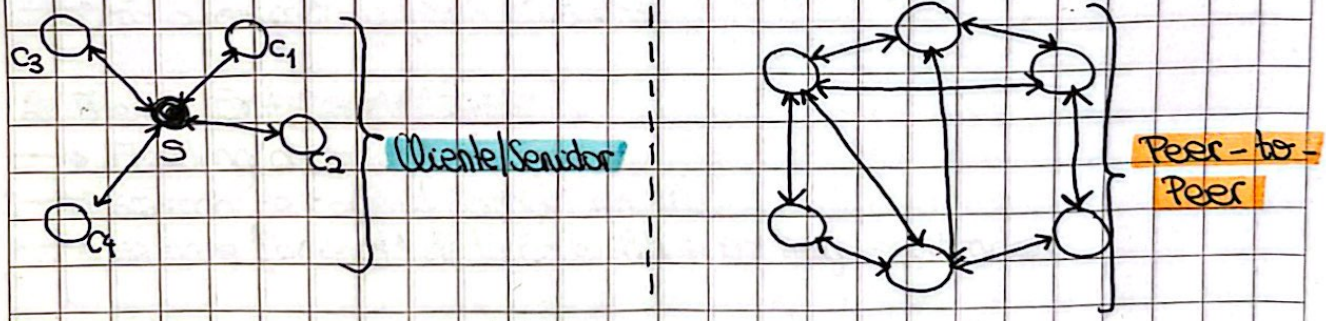
→ cliente/servidor (3-tier), el mismo servidor puede actuar como cliente.

- **Peer-to-Peer**

→ una arquitectura de red distribuida puede ser llamada P2P si los participantes comparten una parte de sus recursos. Ellos son accesibles por otros pares directamente, sin pasar por entidades intermediarias.

→ colaboracion y compartición de recursos a gran escala.

→ potencial de tener millones de nodos y sus participantes son simetricos



→ P2P es altamente escalable

→ 100% descentralizado, no hay cuellos de botella.

→ Tolerancia a fallos (no hay punto unico de fallos).

→ conocimiento local.

- **Overlay:**

→ red de computadores construida sobre otra red.

PROARTE®

• Tipos de overlay P2P
1. Estructurados
2. No estructurados
3. Hibridos.

• P2P estructurados
→ estructura bien definida
→ se utilizan tablas hash distribuidas para su generación.
→ proveen busquedas completas, eficientes y escalables
→ mapeo a espacios logicos por medio de un hash.
→ Problemas: alta latencia de busqueda, poca seguridad, alto dinamismo.

P2P no estructurado
→ Redes no estructuradas
→ conocimiento local, estructura aleatoria, gran autonomia, etc.
→ para conectarme, se llama Bootstrap node, y lo localizo con listas pre-existentes, bootstrap cache.
→ como busco, son incompletas, poco eficientes, completamente descentralizado dificil encontrar contenido no popular, no garantizan resultados.
→ Recursos; se almacenan peers aleatorios, busquedas mas profundas.

• Tipos de busquedas:
1. Flooding o inundacion de mensajes:
→ mensaje enviado a todos los participantes conocidos.
→ poco realista y poco eficiente.

2. Breath First Search (BFS).
→ Flooding controlado
→ sistema de control basado en TTL.
→ se hace "forward" de la consulta si no hay resultados.

3. Iterative Deepning
→ multiples y secuenciales BFS o expanding ring.
→ comienza con busquedas pocas profundas, de no encontrar el recurso itera y prueba mayor profundidad.

4. Directed BFS:
→ se elige un subconjunto de los vecinos y se envia el mensaje.
→ trata de adivinar que vecino es mas probable que responda (Random, resultados con menos saltos, mas vecinos, etc).

PROARTE

4. Local Indexes
→ que archivos tienen mis vecinos y los vecinos de mis vecinos.
→ todos los nodos conocen el radio.

5. Random Walks
→ Diferentes a BFS, mensaje enviado a un vecino aleatorio
→ Si se pilla el recurso retorna un hit, sino el proceso repite hasta TTL max.
→ donde ir se evalua de manera random en cada paso.
→ basado en información es abundante y uniforme.

Super - Peers
→ encontrar información en una red no estructurada es problematico.
→ no hay proceso deterministico para encontrar un recurso en especifico.
→ super-peers son peer con mayores capacidades o roles especiales. (procesamiento, ancho de banda, estabilidad).
→ organizacion jerarquica.
→ Peers regulares conectados a Superpeer.
→ Toda la comunicacion va a través del super peer.

Edge Architectures
→ soluciones en los bordes (edge) de la red.
→ endUsers se conectan a edge servers.
→ Permite optimizar la entrega de contenido.
→ Se aplican tecnicas similares a todo sistema distribuidos.

Problema —> resolver un nombre necesitamos un nodo directorio

— necesitamos seleccionar un contexto donde comenzar la resolución de nombres

Domain Name System

relaciona direcciones IPs con nombres

sino existiese habría que aprender las ips comprometiendo el desarrollo de internet ejemplo: publicidad

estructura jerárquica de árbol

nombres son llamados nombres de dominio ej: udp.cl

contiene componentes de nombre separados por un '.'

no reconoce nombres relativos, concatena dominios por omisión

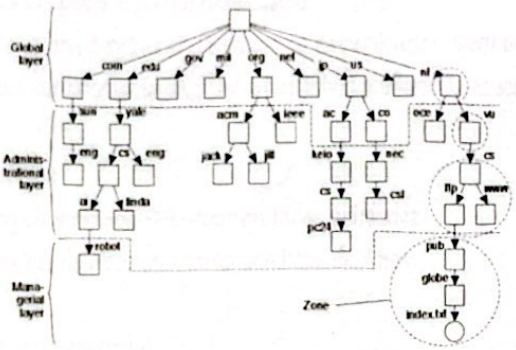
distribuir la resolución de nombres y el espacio de nombres de nombres sobre múltiples máquinas

tres niveles :

Global Level— directorios de alto nivel manejados por varias administraciones

Administrational Level — directorios de nivel intermedio

Managerial Level —directorios de bajo nivel manejados por una administración (local DNS)



¿Cómo funciona?

Participantes :

DNS Resolver — recibe consultas de los clientes por medio de aplicaciones como web browser — resolver es responsable de realizar requests adicionales para satisfacer la consulta del cliente

Root Nameserver — primer paso es la traducción de nombres a IP addresses — entrega información más especifica de una localización

TDL Nameserver — paso siguiente en la búsqueda de una IP address y almacena la última porción de un hostname ejemplo example.com almacena '.com'

Authoritative Nameserver — ultima parada en la búsqueda — accede al registro y devuelve la ip address al host solicitado y lo envía de vuelta al DNS resolver

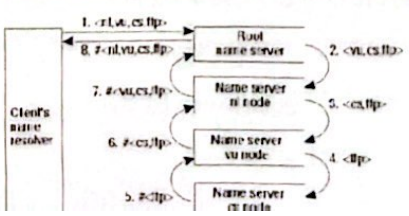
DNS Lookup

- usuario tipea 'example.com' en el web browser -- consulta (querrá) viaja por internet -- llega al DNS resolver

resolver envía la query a DNS root nameserver -- root server responde a el resolver con una dirección de un TLD DNS server -- almacena la información de estos dominios -- dada la query 'example.com' la request es redirigida a .com TDL -- resolver realiza la request a .com TLD -- TLD server responde con el IP address del dominio nameserver -> example.com -- resolver envía una consulta al dominio nameserver -- ip address es devuelto al resolver desde el nameserver -- DNS resolver responde al web browser con la IP address del dominio solicitado inicialmente

Tipos de queries:

Recursivo— DNS resolver client se comunica con un DNS server tipicamente un DNS recursive resolver — responde con el nombre o error en caso que no se encuentre

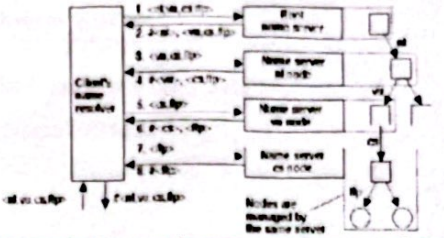


Recursive DNS resolver

- máquina que responde a una request recursiva y se encarga de gestionar las request para obtención el DNS récord
- realiza una serie de request hasta encontrar el authoritative DNS nameserver para el récord solicitado (o entrega el error si no lo encuentra)

Iterativo: — DNS resolver client permite al DNS server responde con la mejor respuesta que encuentre — sino encuentra el nombre, se retorna una referencia a un dominio más bajo

— proceso continua hasta tener una respuesta, error o timeout



No Recursivo — DNS resolver client se comunica con un servidor de nombre buscando un registro y este es Autoritative Nameserver o bien se encuentra en el cache de este

técnica ampliamente utilizada para reducir el consumo de ancho de banda

Resolución recursiva permite a cada servidor aprender acercada donde ubicar la consulta — ideal para cache

en approach iterativo el cache es necesario pero a nivel cliente

en general es de menor costo que una iterativa en términos de comunicación, pero es más costosa en computo para el servidor

demanda más recursos, por eso servidores de nivel superior(global) prefieren búsqueda iterativa

Attribute-Based naming

servicios de directorio tradicionales

problema — costoso, matching de atributos -> buscar en todas entidades

solución— implementar un servicio de directorio como una base de datos

Relojes físicos

cristal +2 registros (contador y almacenado)

cada oscilación decrementa el contador, al llegar a cero se genera una interrupción y reinicializa al valor del registro almacenador

cada interrupción es tic de reloj

al iniciar el computador se solicita fecha y hora que se traduce a un número de tics

proceso locales no les afecta reloj en cuanto avance normalmente

a pesar de que la frecuencia del cuarzo es estable, es imposible garantizar que todos los cristales de los computadores oscilen a la misma frecuencia

- intensidad de corriente y corte del cristal

problema —> gradualmente los relojes se desajustan , diferencia entre relojes de las máquinas es conocido como deriva del reloj

¿Cómo medimos el tiempo?

— intervalos entre 2 tránsitos del sol (punto más alto)

— intervalo rotación tierra no es constante

— rotación se hace más lenta —> día más largo

TAI — international atomic time

solución— introducir saltos

NIST —> provee UTC por medio de una estación de radio de onda corta (WWV)

Sincronización

Algoritmo de Cristian

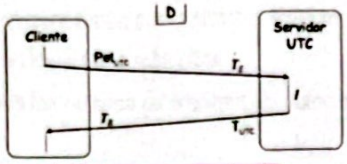
— pensado en entornos con servidor de tiempo sincronizado con UTC y las estaciones desean sincronizarse con el

— cada máquina pregunta a el time server cada : $\delta/2\rho$ segundos (δ: Segundos de diferencia; ρ: máxima deriva) la hora

— servidor de tiempo responde con su hora actual Tutc— cuando recibe el cliente debe actualizar su reloj

En Sistemas Distribuidos es Importante Mantener Sincronizados los Equipos	Dos Equipos con $V_d = \rho$ Opuesta Después de $\Delta t \Rightarrow$ Dif. hora = $2\rho\Delta t$ Para mantener dif. máxima = δ seg. Sincronización cada $\delta/2\rho$ seg.

si la hora actual es posterior a UTC, debe ralentizar su reloj, en cada interrupción se añaden 9 ms
si la hora actual es anterior al recibido , lo adelanta al UTC o lo acelera progresivamente

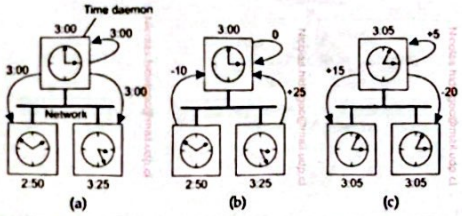


Algoritmo de Cristian → Error promedio = $\frac{(T_1 - T_0)}{2+1}$

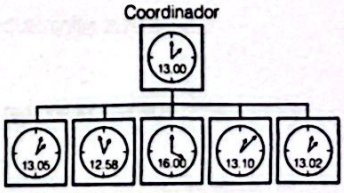
Problema —> punto unico de fallo— es centralizado
solucion —> varios servidores UTC, clientes considera la primera respuesta que llega
problema—> relojes erróneos y propagación de error, aun se puede tener consenso si $N > 3f$

Algoritmo de Berkeley

— pensados para ambientes sin acceso a UTC
— principio radica en mantener sincronizadas las máquinas a una misma hora — no es necesario sincronizar a tiempo real
— selección de coordinador o servidor de tiempo

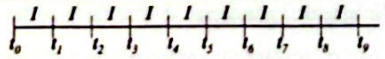


Promedio tolerante a fallos —> solo utiliza un subconjunto de los tiempos recibidos para calcular el promedio — los que no varían en más de un valor Δ. El resto se consideran relojes fallados
si falla el coordinador se selecciona otro — no pregunta hora, no envía notificaciones, etc



ALGORITMO DISTRIBUIDO

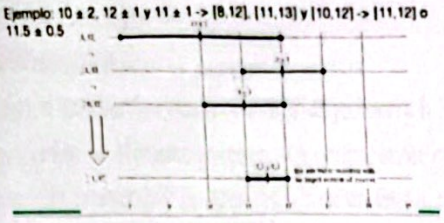
1. Se deben conocer los tiempos de propagación
2. Todos los equipos deben sincronizarse periódicamente



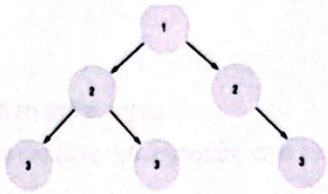
3. Al comienzo de cada intervalo, todos difunden su hora local
4. Esperan un tiempo para recibir la hora de los demás equipos
   - Se desechan los valores extremos
   - Con todos los valores recibidos se calcula la desviación media
5. Con la desviación media se actualiza el Tiempo Local.
6. Todos los nodos tienen la misma hora.

Network time protocol (NTP)

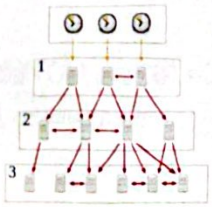
generado para proveer servicio de tiempo en internet
tiempos de transmisión en internet son variables
usa técnicas estadísticas para filtrar los tiempos de calidad
utiliza el algoritmo de marzullo —> se utiliza par seleccionar fuentes de tiempo, relojes, para sincronizar tiempo entre varios mejores , basado en un origen o fuente único

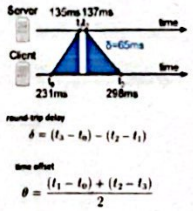


servicio debe ser escaladle para lidiar con constantes peticiones de sincronización
autenticación de servidores para proveer un servicio seguro (maliciosos o accidentales)
red de servidores en internet —> servidores conectados en una estructura jerárquica o sub-red de sincronización
niveles de jerarquía son conocimos como strata o stratum
errores son mayores en los stratos más altos
se considera igualmente los tiempos de transmisión como métrica de calidad
strata 2 sincroniza con strata 1, strata 3 sincroniza con strata 2, etc



3 modos de sincronización:
Client/Server: servidor chequea uno o más servidores para sincronizar
Symmetric: para stratos bajos de la arquitectura— se utilizan entre ellos para generar backup mutuo de referencias
Broadcast/Multicast: se usa cuando se requiere poca precisión— uno sincroniza y notifica a todo el resto





se calculan los tiempos de propagación aplicando el concepto de dispersión (suma de dispersiones servidor-root)
se aplica una fase para la modificación de la frecuencia de los relojes en base a las derivas observadas
Delay — indica tiempo de ida y vuelta en segundos desde que el cliente inicia la comunicación hasta que recibe la respuesta
Offset — indica la diferencia de tiempo, en segundos entre el reloj del cliente y el de la fuente
Jitter — indica el error máximo en segundos , del reloj del peer en relación al reloj local y la ruta de red que los separa
Poll — indica el intervalo de contacto
When— indica el número de segundos transcurridos desde el último contacto
T(type)— indica el tipo de sincronización , puede ser :
- l: local por ej: wwvb o gps
- u: unicast
- m: multicast
- -: desconocido

Reach— registro de cambio que se utiliza para determinar el estado de accesibilidad del peer — un peer se considera accesible si al menos un bit en este registro esta establecido en 1

Relojes lógicos

no es necesario sincronizar procesos que no interactúan, su falta de sincronización no será observable , deben concordar con el orden en que ocurren los eventos

Precedencia — ocurre antes de .

a~b — a ocurre antes que b

todos los procesos concuerdan que a ocurre y luego ocurre b

1) si a y b son eventos en el mismo proceso, y a ocurre antes que b entonces a ~ b es cierto

2) si a es un evento enviado por un proceso y b es un evento recibido por otro, entonces a ~ b es cierto

la relación "ocurre antes que " — es transitiva

si a ~ b y b ~ c , entonces a ~ c

Concurrentes — no se puede saber el orden de como ocurrieron (~ o ‖)

evento a tiene un valor de tiempo C(a) en el que todos los procesos concuerdan

estos tiempos deben cumplir a ~ b, es decir C(a) < C(b)

el tiempo de reloj C , siempre debe avanzar y nunca retroceder

problema —> actualizaciones deberían realizarse en el mismo orden en ambas copias

solucion —> totally ordered multicast —> multicast donde todos los mensajes son entregados en el mismo orden , como relojes lógicos de lamport

Multicast entre procesos

cuando un proceso recibe un mensaje lo pone en su cola local ordenado de acuerdo a su timestamp

receptor envía un ACK a los otros procesos

siguiendo lamport , si el timestamp del mensaje recibido tiene que ser mejor al ACK

cada proceso tendrá la misma copia de la cola

proceso pasa el mensaje a la aplicación si : mensaje si esta en el reader de la cola, ACKs de todos los procesos han llegado para ese mensaje — se asume comunicación confiable

Reloj lógico —> a ~ b, entonces C(a) < C(b), pero C(a) < C(b) no implica que a ~ b

Reloj de vector —> CV(a) < CV(b), entonces a ~ b

Vectores :

cada proceso Pi tiene un arreglo VCi [n] con n numero de procesos involucrados

VCi [j] es el número de eventos que el proceso Pi sabe que se han llevado a cabo en Pj

cuando un proceso i envía un mensaje comparte su VCi

Algoritmo:

- Inicialmente, $V_i[j] = 0$, $\forall i, j$
- Evento local o envío de mensaje en p_i: $V_i[i] = V_i[i] + 1$
- Cuando p_i envía un mensaje m a p_k, p_k incluye el valor de su vector de tiempos, V_m. Al recibir dicho mensaje, p_k actualiza su vector de tiempos de la siguiente manera:
  (1) $\forall i: V_k[k] = max(V_k[k], V_m[k])$. (2) $V_k[i] = V_k[i] + 1$

Reloj Vectorial

existirá una relación de causalidad entre 2 eventos a y b si solo si VC(a) < VC(b) o VC(b) < VC(a), sino son concurrentes

VCi < Vci' si solo si—> VCi [j] =< VCi' [j] para todo j y existe un j / VCi [j] =< VCi' [j]

Exclusión Mutua

procesos deben cooperar de manera concurrente

recursos compartidos pueden ser accedido de manera concurrente

para asegurar inconsistencias o corrupción de información

Requisitos:

Seguridad— proceso puede ejecutar el recurso compartido (región crítica)

Viveza — se debe garantizar que un proceso que desea acceder a el recurso haga ingreso en un momento dado

Orden — garantizar orden causal, si un proceso P1 envía un mensaje a P2 y ambos requieren un recurso P1 deberá accederlo antes que P2

garantizan multiples procesos distribuidos no se interfieran y logren cooperar
2 tipos : Token-Based y Permission-Based

Token-Based

exclusión mutua se realiza por medio de un token — token único quien lo tenga puede hacer uso del recurso compartido— cuando termina, el token es pasado al siguiente proceso, si el proceso siguiente no esta interesado en usar el recurso, se pasa inmediato al siguiente
propiedades:
aseguran de manera fácil que los procesos van acceder el recurso compartido, no hay "starvation" o inanición
procesos se esperan unos a otros por el token por lo que no hay deadlocks
problema—> pérdida del token (caída del nodo), se debe iniciar un proceso complejo de creación de un nuevo token

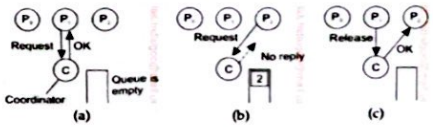
Permission-Based

proceso solicita permiso a otro dos procesos— 2 mensajes: mensaje solicitud y mensaje autorización
Algoritmos de exclusión mutua:

Centralizados — servidor único

- unico proceso coordinador
- si algún proceso desea acceder al recurso compartido, envía mensaje a coordinador — recurso esta libre, entrega acceso respondiendo con el permiso— al recibir el mensaje con el permiso, proceso puede acceder al recurso
- solo un proceso a la vez puede acceder el recurso compartido
- se respeta el orden de las solicitudes
- ningun proceso espera por siempre

solo 3 mensajes por recurso — request, grant , release
problemas : punto unico de fallo, si falla todo el sistema también — luego de realizar una petición, el proceso no puede identificar un coordinador caído de un permiso denegado — cuello de botella en un sistema de gran escala



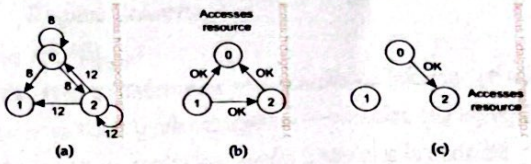
Algoritmo Descentralizado— usando un sistema P2P

—> solución : descentralización — algoritmo de voto sobre un DHT, recurso replicado n veces cada uno con su coordinador , se requiere mayoría de m > n/2 coordinadores, m números de mensajes mayoría y k intentos (back-off)
cuando el coordinador no concede acceso, el coordinador notifica
elimina cuello de botella y punto unico de fallo

Algoritmos Distribuidos — sin topología o usando anillo lógico

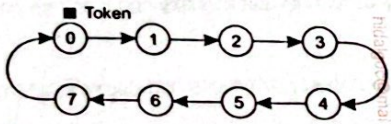
algoritmos de marcas de tiempo — mejora el algoritmo de lamport — se requiere ordenamiento total
Algoritmo —>
proceso que desea un recurso compartido genera un mensaje que contiene —> nombre del recurso ; numero del proceso; reloj lógico actual
envia mensajes a todos los procesos incluido el
no hay perdida de mensajes
Proceso recibe un mensaje —> receptor no esta accediendo al recurso y no quiere accederlo -> envía permiso ok
receptor ya accedió al recurso -> no responde y lo pone en cola
proceso también quiere acceder al recurso y aun no lo ha hecho, compara timestamp del mensaje con el suyo — si es menor el del mensaje— envía permiso ok, sino pone en cola el mensaje y no responde
debe esperar todos los permiso para acceder a el recurso
cuando termina envia ok a todos los procesos en su cola y los elimina
garantiza que todo accedan al recurso y que no se produzcan deadlocks

as híbridas :



problemas : caida de nodos, la no respuesta es considerada como una negación de acceso
solución— responder siempre o mayoría de votos
— manejo de multicast en cada proceso, mantener la lista de membership
— más complejo que el centralizado, n puntos fallo y con más tráfico

Token-Ring
anillo lógico de equipos, cada uno con una posición dada según ip
cada nodo conoce la dirección de su vecino siguiente
al arrancar, primer proceso recibe el token el cual ira circulando por el anillo
cuando proceso k tiene el token, debe transferirlo al proceso k+1 por un mensaje
si el proceso desea hacer uso del recurso y tiene token, puede acceder a el reteniendo el token — cuando libera el recurso pasa el token



problemas: perdida de token, cada de nodos

Repaso Solemne 2

Propiedades para hacer escalar al DNS
- particionar : particiona por dominios y subdominios —> guardamos información relativa a una URL hace una relación texto a una dirección IP
- replicas : niveles superiores —> globales y administrativos —> niveles que agregan mayor cantidad de carga
- hash/cache : nivel cliente -> resolver — recibe la consulta y resuelve la consulta

tablas de hash distribuidas (DHT) —> P2P estructurado
como logra:
- alta disponibilidad —> cada par se alojaba en un espacio numérico — porque al aplicar un hash sobre la dirección ip: port
— se logra por el proceso de mapeo donde rompe localidad y eso evita particionamiento
- escalabilidad —> explota la calidad en las entradas de la tabla —>> número reducido de entradas —> se puede localizar a cualquiera
- balance de carga —> hash -> se distribuyen uniformemente los recursos indexados —> ejemplo hash sha-1

Problema de sincronización— se puede abordar con relojes físicos y lógicos
¿por que al utilizar un reloj fisico hay problemas de sync?
- corriente y calidad cuarzo —> difieren de par a par siempre hay un efecto de deriva

para determinar los tiempos— los tiempos divergen por eso no se pueden usar en gran escala porque al divergir los tiempos el promedio del tiempo no será el mismo para cada uno
solución —> NTP a gran escala se tiene mediciones por cada nodo y peticiones de los peer que están participando dentro de lo mismo

metodos recursivos DNS -> se elige el iterativo sobre el recursivo para evitar la sobre carga del TDL

Orientada a la computación de alto rendimiento (HPC)

Clusters

-> nodos homogéneos conectados en una red rápida

-> Hardware homogeneo

-> utilizados tradicionalmente para aplicaciones paralelas, programa único corriendo en paralelo en varias máquinas

Grid

-> nodos heterogéneos conectados a una red compartida bajo varias administraciones

-> organización virtual con recursos para compartir

-> computación y almacenamiento

-> colaboración de diferentes recursos

Cloud público (AWS, azure, google cloud, etc)
- no hay información/garantías en ubicación, multi-tenacy, rendimiento real

Cloud Privado
- permite monitoreo y toma de mediciones
- no hay control sobre la configuración de la infraestructura

Grid -> completamente reconfigurable
- se pueden modificar todos los niveles (VM, OS, Network..)

Peer-to-peer -> nodos heterogéneos conectados a una red compartida completamente abierta
- comunicación directa
- sistema abierto
- completamente descentralizado
- tipos: estructurada, no estructurada, híbrida
- ejecución de programas que no se pueden ejecutar de manera centralizada, restricciones de computo
- descomposición de la aplicación en micro-tareas

Sistemas de información
- sistemas principalmente para el manejo e integración de funciones del negocio
- Enterprise information system (EIS) : cualquier sistema de información que mejora las funciones de un proceso de negocio
- procesadores de transacciones : -> primitivas especiales -> atómicas, consistentes, aisladas, durables (ACID)
- componentes o servicios (SOA)

Sistemas Pervasivos
- no hay control centralizado
- incluyen la componente la movilidad
- características especiales : energía, procesamiento, almacenamiento, etc
    - home systems, health , smartcities

Arquitecturas distribuidas -> una buena organización es clave para alcanzar las propiedades deseadas

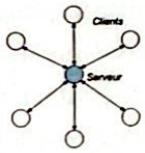
Arquitectura del sistema -> si consideramos donde colocamos los componentes de software tenemos:
- arquitecturas centralizadas : cliente/servidor
- arquitectura descentralizada: peer-to-peer
- arquitecturas híbridas : edge

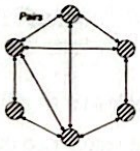
Cliente/Servidor

- dos sub-sistemas o procesos (2-tier)- cliente/servidor simple
  clientes: proceso que solicita un servicio
  servidores: proceso que implementa un servicio en especifico
- aplicación es modelada como un conjunto de servicio proveídos por los servidores y un conjunto de clientes que usan los servidores
- paso de mensajes (interacción request-reply)
- escalamiento vertical
- cuello de botellas posibles
- modelo seguro
- dos maquinas (2-tier) y 3 capas (3-levels): interface application y database
- tendencia a mantener el procesamiento en el lado del servidor
- cliente/servidor (3-tier) -> multi-tier: mismo servidor puede actuar como un cliente

Peer-to-Peer

- arquitectura de red distribuid puede ser llama P2P si los participantes comparten una parte de sus recursos . Ellos son accesibles por otros pares directamente, sin pasar por entidades intermediarias
- colaboración y compartición de recursos a gran escala
- participantes son simétricos





C/S vs P2P

- altamente escalables
- 100% descentralizado , no hay cuellos de botella
- auto-organizados , participantes autonomos
- tolerancia a fallos (no hay un único punto de falla)
- conexión y participantes no confiables (abierta y anónima)

Overlay-> red de computadores construida sobre otra red

P2P Estructuradas

- construidas con proceso deterministico
- tienen estructura bien definida
- se utilizan tablas de has distribuidas para su generación
- capaces de escalar a los cientos de millones de participantes
- proveen de búsquedas completas
- búsqueda eficiente y escaladle orden logarítmico (los N)
- rapeo a espacio lógico por medio de has tipo sha-1 o md5
- ruteos diferentes dependiendo de la geometría del overlay
- Key Based Routing protocols (KBR)

-> tree , hypercube, butterfly, platoon trees

Chord

- organización tipo anillo -> 0 ... 2^n
- cada peer posee un identificador generado con un SHA-1 (consistent hash)
  -> node id: SHA(IP Address + puerto)
  -> recurso id (key): SHA(recurso)
- auto-organizado

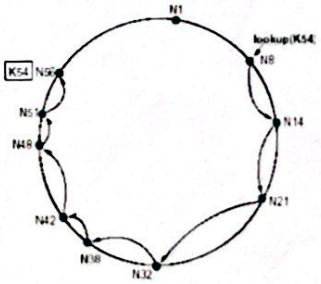
* Mantenimiento caro ($\log N$)^2

-ruteo de mensajes de tipo árbol
- asignación de recursos -> sucesor de la key
- asignación en sentido del reloj (uni-direccional)
- peer entra (join) -> recibe recursos -> se hace responsable de un grupo de recursos (sucesor)
- peer deja la red (leave) -> entrega recursos al nuevo sucesor

Chord vs Churn

Churn -> join/leave crean inconsistencias en tablas
- reparación es costosa , protocolo de estabilización
- se detectan caídas por hear beats messages
- guarda replicas en S sucesores

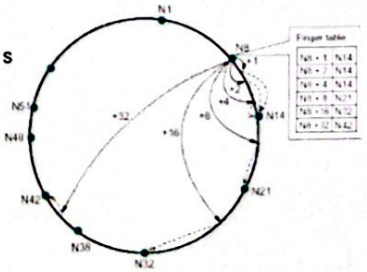
¿Cómo participamos?
- identificador en el mismo espacio, se rutea identificador
- contacta al sucesor con su identificador
- consistencia de tablas
  —> sucesor entrega información de su antecesor (entradas tabla)
  —> sucesor agrega a un nuevo peer como antecesor
- distribuye los recursos de los cuales es responsable el nuevo peer



¿Cómo se busca?
- cada peer conoce su sucesor en el anillo
- modelo simple de búsqueda -> seguir links a sucesores

¿Cómo mejorarla?
- agregar información
- fingertable(DTH)-> tabla que contiene información acerca de log m peer en la red -> permite saltar peers y acercarse más rápido al peer responsable

objetivo -> encontrar el sucesor de la clave —> métrica: cercanía entre identificadores



Pastry

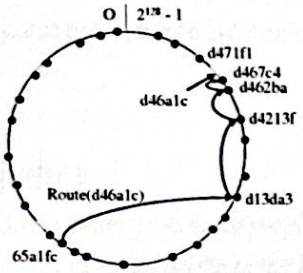
- organizado en anillo
- topologia ruteo Plaxon-Tree
- ruteo basado en prefijo
- identificador de 128 bits generado con hash SHA-1
- espacio 0-2^128
- ruteo log N saltos al destino

Pastry Leafset

- pastry guarda L peers llamados leafsets nodes
- para un peer X, su leafset esta compuesto -L/2 y L/2 peers en el anillo
- para balancear cargas , tolerancia a fallo, replicas, multi-path search

Tabla de ruta Pastry

- organizada por prefijo
- fila i contiene aquellos ids de peer que comparten un prefijo de tamaño i
- tuteo por prefijo asegura acercarse sucesivamente a la clave buscada
- mas prefijo en común , más cerca del destino



—> Routing Pastry

Tabla de vecinos

- peer cercanos en términos de lantencia
- útiles para generar búsquedas diversas en caminos
- se genera -> ping para estimar latencia o simplemente IP

Overlays estructurados

- búsqueda puntual es un problema para las aplicaciones
- se resuelve con Prefix-tree (PHT), multianillos, etc
- se puede generar un overlay que preserve la localidad -> LSH

P2P no estructurado

- estructura aleatoria
- conocimiento local
- métodos de búsqueda basado en inundación de mensajes o flooding
- gran autonomia

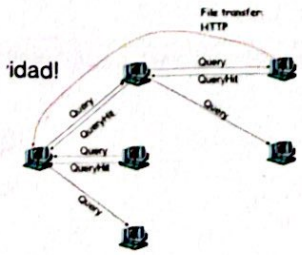
¿Cómo me conecto?

- participante llamado Bootstrap note
- lo localizo a través de listas pre-existentes, bootstrap cache

problema para limitar conexiones con el fin de balancear la carga

¿Cómo busco?

- incompletas
- poco eficientes, muchos mensajes
- comparación de string
- completamente descentralizada
- efectivo en contenido popular
- búsqueda no garantiza resultado



Recursos

- se almacenan en peers aleatorios
- recursos pueden quedar lejos de los peers —> max TTL=7 en gnutella 0.4
- búsquedas más profundas , más mensajes
- caching -> problema de investigación

Tipos de búsqueda

Flooding o inundación de mensajes

- mensaje enviado a todos los participantes conocidos
- poco eficiente
- calidad resultados vs cantidad mensajes

Breath First Search (BFS)

- flooding controlado
- sistema de control basado en TTL o routing hops
- si no hay resultados -> forward
- proceso se repite TTL veces
- mensajes duplicados, gran overhead

Iterative Deeping

- multiple y secuenciales BFS o expanding ring
- comienza con búsquedas poco profundas
- al no encontrar el recurso, itera y prueba mayor profundidad

—> guarda por un tiempo la consulta evita mensaje repetidos

Directed BFS

- se elige un subconjunto de los vecinos y se envía el mensaje
- trata de adivinar que vecino es más probable que responda —> random, resultados con menos saltos, + mensajes recibidos, + vecinos, - latencia, etc

Local Indexes

- que archivos tienen mis vecinos y los vecinos de mis vecinos
- todos los nodos conocen el radio
- problemas de memoria para mantener listas
- construir la lista, extra overhead

Random Walks

- mensajes enviado a un vecino aleatorio
- si se pilla el recurso retorna un hit, si no proceso se repite hasta TTL max
- se evalúa donde ir de manera random en cada paso
- ciego y sin memoria
- basado en que la información es abundante y uniforme
- utiliza un walker
- disminuye el numero de mensajes
- aumenta la latencia al esperar respuestas

K-Random Walks

- multiples random walks paralelos
- disminuye latencia
- variantes : —> APS

—> id- based , enviable solo al que no ha ruteado consulta

—> two-level random walk, k1 random walkers por un TTL T1, luego k2 random walkers por un TTL T2

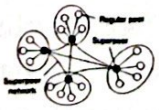
Condiciones de termino :

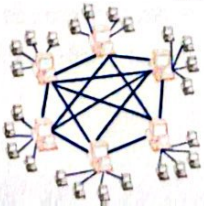
TTL-> dificil setear el parámetro , red dinámica no ayuda

Chequeo resultados -> se chequea con el nodo que consulta frecuentemente , normalmente cada 4 saltos

Super-Peers

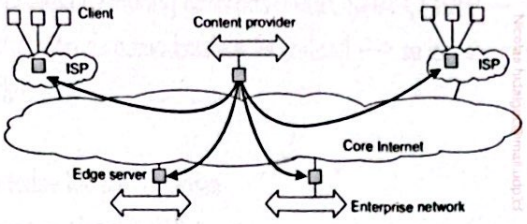
- super peers son peer con mayores capacidades o roles especiales -> procesamiento, ancho de banda, estabilidad
- organización jerárquica
- peers regulares conectados a super peer (SP)
- toda comunicación va a través del super peer
- relación estable con super peer
- SP larga sesión, alta disponibilidad





Edge Architectures

- soluciones en los bordes (edges) de la red
- end users se conectan a edge servers
- permite optimizar la entrega de contenido
- se aplican técnicas similares a todos sistema distribuido -> replicas en otros edge servers ejemplo: web pages



procesos no pueden compartir recursos a menos que los nombren consistentemente

Local Vs Distribuido

Foco en lo distribuido —> implementación del servicio de nombres es distribuido sobre múltiples máquinas — juega un rol importante en la eficiencia y escalabilidad del sistema

Naming— identificar

cadena de letras (storing) que hace referencia a una entidad ejemplo: hosts, impresoras, discos, archivos, procesos, web pages, etc.
entidades pueden ser operadas ej: imprimir
primero hay que accesarla —> dirección y mecanismo para localizar un recurso
se pueden tener múltiples direcciones ejemplo: combinación ip/puerto

Identificador vs Nombres

requiere que sean únicos, identifica a lo más una entidad

cada entidad es reverenciada por a lo más un identificador

identificador siempre hace referencia a la misma entidad
reduce la ambigüedad para acceder a una entidad

problemas -> entidad cambian , nombres no son únicos

URI: Uniform Resource Identifiers

son uniformes y coherentes— permite el procesamiento de los nombres por softwares comunes ejemplo: browser
compuesta por 3 partes : esquema, nombre de máquina donde se aloja el recurso, nombre del recursos en forma de ruta(path o ruta)
subtipos: **URL (uniform resource locators) y URN(uniform resource names)**

URL:

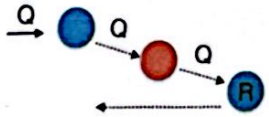
provee información acerca de un recurso y como localizarlo — ejemplo: http://www.google.cl
eficiente para accesar recursos , pero no es eficiente ante eliminación o cambio del recursos

URN:

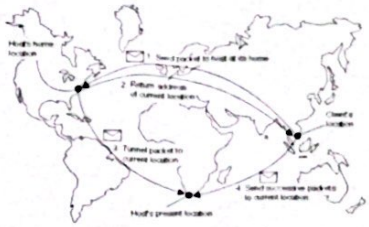
nombres de recursos puros, no para localizar — identifica un mensaje particular usando el "Message-id"
distingue este mensaje de otros, pero no provee la dirección del mensaje en ningún almacenamiento
proceso o mecanismo permite la asociación de un nombre con una dirección —> name-to-address
3 tipos de sistemas : Flat Naming, Structured Naming, Attribute-based naming

Flat Naming

soluciones simples —> broadcast and multicast , forwarding pointers
home based solutions
DHT's
Identificador Random —> Cadena de storing generada (Hash, SHA-1, MD5) — sin estructura
identificador no provee información de como localizar la entidad —> se localiza por Lookup
espacio de búsqueda limitado
Broadcast y multicast



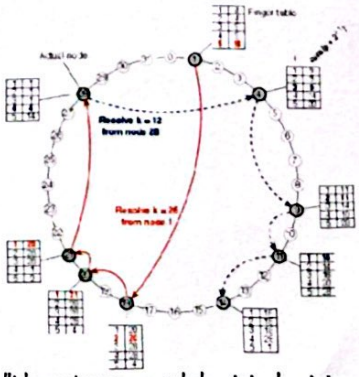
—> se envía un mensaje a todos los participantes
—> no escala más allá de una red local (LAN)
—> todos los procesos deben estar escuchando
Multicast reduce el número de mensajes y mejora la escalabilidad sin embargo sigue teniendo escalabilidad limitada
Forwarding Pointers —> cuando una entidad se mueve dejando una referencia hacia su nueva ubicación , cuando se localiza se hace el update de la ubicación
Home-Based Approaches: Mobile IP
- home mantiene donde la entidad se encuentra
- home address de la entidad es registrado en un servicio de nombres
- clientes contactan el servicio de nombres y luego van a la ubicación de la entidad
- chequea registro local — va al home address si falla el local



Problemas? — home address es fijo —> sobrecarga cuando la entidad cambia frecuentemer de su lugar, escalabilidad limitada

como buscamos?

DHT's— Distributed Hash Tables
Chord —> nodos en espacio lógico 2^m
cada nodo tiene id random único (m-bits) m=160 —> 1,46+48 ids
cada entidad tiene asignada una key de m-bits
entidad id es responsable de la key k si — id >= k (sucesor)
problemas ? -> proximidad semantica , latencia
soluciones -> Proximity Aware Overlay, Proximity Routing



Structured Naming — sistema cuyo identificador posee una estructura
"human-friendly" son estructurados y compuestos de varias partes —> nombres, apellido paterno, apellido materno — subdominio.dominio
espacio de nombre representado como un grafo cíclico dirigido (DAG)
estructura jerárquica — existe un único primer nivel que se expande abarcando toda la red, el **root**
red es dividida en una colección de dominios — cada dominio es dividido en sub dominios de menor tamaño
hojas —> dominios locales
puede crecer de manera infinita ejemplo: /etc/passwd