Erick Vargas

Acordeón 1er parcial

Contents

1

Escalabilidad (1.4.4)

Los sistemas distribuidos no tienen un tamaño definido, pueden ser muy pequeños o enormes.

Un sistema es escalable si y solo si conserva su efectividad cuando hay un incremento de usuarios y/o recursos.

Los sistemas distribuidos escalables tienen varios retos:

1. **Control del coste de recursos físicos:** Si crece la demanda de un recurso y se extiene el sistema, el costo de hacer esto debería ser razonable. Para que un sistema sea escalable para cualquier cantidad n de usuarios la cantidad de recursos físicos necesarios que soporte este sistema debe ser $O(n)$

2. **Control de las pérdidas de prestaciones:** Si crece la cantidad de recursos que tiene un sistema deben mantenerse las prestaciones es decir: tiempo en el que se accede a un dato $O(log(n))$, si tenemos esta medida la pérdida de prestaciones será mínima. (Se usan sistemas jerárquicos)

3. **Prevención de desbordamiento de recursos software:** Un ejemplo sencillo es el número de cantidades IP el cual empezó con direcciones de 32 bits pero a principios del año 2000 estaban prácticamente agotadas estas direcciones por lo cual se optó por utilizar en su lugar direcciones de 128 bits.

4. **Evitación de cuellos de botella de prestaciones:** Los algoritmos que se utilizan deben ser descentralizados. (Distribuir información en varios servidores con el fin de agilizar el acceso a recursos importantes)

1.1 Preguntas

1. ¿El soporte de la cantidad de recursos físicos al escalar un sistema distribuido debe ser de una complejidad?

- $O(n)$
- $O(log(n))$
- $O(nlog(n))$
- $O(n^2)$

2. ¿Para evitar el perder prestaciones el tiempo de acceso a algún dato en un sistema distribuido debe ser almenos de una complejidad?

   - $O(n)$
   - $O(log(n))$
   - $O(nlog(n))$
   - $O(n^2)$

2

Tratamiento de fallos (1.4.5)

Si hay fallas en software o hardware los sistemas deberían de mantenerse en funcionamiento. En un sistema distribuido los fallos deben ser parciales, es decir si algún componente falla el resto debe mantenerse en funcionamiento.

- **Detección de fallos:** es posible detectar algunos fallos haciendo uso de algunos algoritmos, por ejemplo utilizando el checksum
- **Enmascaramiento de fallos:** algunos fallos que han sido detectados pueden ocultarse o atenuarse. Dos ejemplos son

  1. Mensajes retransmitidos cuando hay fallas en recepción
  2. Se realiza una copia de los archivos en discos, si uno falla el otro sirve de respaldo

  En el caso de atenuar fallos eliminar mensajes corruptos es un ejemplo de ello. No siempre es efectivo.
- **Tolerancia de fallos:** En sistemas enormes es dificil cubrir todos los fallos por tanto los clientes ”sufren” un poco esto. Los clientes se diseñan para tolerar ciertos falloslos cuales los usuarios también deberán tolerar (generalmente)
- **Recuperación frente a fallos:** Supongamos que un servidor se cae, los datos del servidor deben de poder recuerarse (roll back) a un estado anterior.
- **Redundancia:** hay fallos que pueden tolerarse si se utilizan componentes redundantes, por ejemplo:

  1. Dos rutas diferentes entre dos routers
  2. Sistemas de nombres de dominio replicados en servidores diferentes
  3. Bases de datos replicadas en varios servidores para garantizar que los datos siempre seran accesibles

2.1 Preguntas

1. ¿Qué debería ocurrir en caso de que un componente de nuestro sistema distribuido deje de funcionar?

    - **Todo el sistema se detiene**
    - El sistema debe mantenerse en funcionamiento