Replicación y consistencia

Tema 2

Detectores de fallos

Índice

1. El problema de la pertenencia a grupo.
2. Concepto de detector de fallos.
3. Clasificación.
4. Aplicabilidad.

Bibliografía

[CHT+96] T. D. Chandra, V. Hadzilacos, S. Toueg, B. Charron-Bost: “On the Impossibility of Group Membership”, 15th ACM Annual Symposium on Principles of Distributed Computing, pgs. 322-330, New York, USA, 1996.

[CKV01] Gregory Chockler, Idit Keidar, Roman Vitenberg: “Group Communication Specifications: A Comprehensive Study”, ACM Computing Surveys, 33(4):427-469, diciembre 2001.

[Cris91] F. Cristian: “Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems”, Distributed Computing, 4(4):175-188, 1991.

[CT96] T. D. Chandra, S. Toueg: “Unreliable Failure Detectors for Reliable Distributed Systems”, Journal of the ACM, 43(2):225-267, marzo 1996.

[JFR93] F. Jahanian, S. Fakhouri, R. Rajkumar: “Processor Group Membership Protocols: Specification, Design and Implementation”, 12th IEEE Symposium on Reliable Distributed Systems, pgs. 2-11, Princeton, New Jersey, USA, octubre 1993.

Índice

1. **El problema de la pertenencia a grupo.**
2. Concepto de detector de fallos.
3. Clasificación.
4. Aplicabilidad.

1. El problema de la pertenencia a grupo

- En un sistema distribuido suele ser útil saber qué máquinas o procesos están activos en cada momento.
- Pero no basta con que cada proceso tenga cierta noción acerca de esto. Convendría que todos ellos compartieran dicha información.
  - Para ello se debe ejecutar cierto algoritmo que asegure el acuerdo entre todos: consenso.
  - Son los servicios de pertenencia a grupo.

1. El problema de la pertenencia a grupo

- Los primeros servicios de este estilo fueron especificados por Flaviu Cristian [Cris91] para sistemas distribuidos sincrónicos.
- Posteriormente se publicaron otros trabajos para sistemas asíncronos, como por ejemplo [JFR93].
- A mitad de los 90 [CHT+96], se demostró que este servicio no era implantable en sistemas asíncronos, al igual que tampoco se puede resolver el problema de consenso en este tipo de sistemas.

1. El problema de la pertenencia a grupo

- A pesar de lo anterior, en cualquier sistema moderno de comunicación de grupos, suele existir un servicio de pertenencia [CKV01].
  - Dichos servicios de pertenencia se apoyan en un resultado teórico sobre detección de fallos [CT96]: no son perfectos, ni lo podrán ser en un entorno asíncrono, pero sí utilizables si los algoritmos que los necesiten conocen sus limitaciones.

1. El problema de la pertenencia a grupo

- Ventajas de un servicio de pertenencia:
  - Permiten que todos los componentes tomen acciones uniformes ante una caída o reincorporación de un proceso/nodo.
  - Simplifican los algoritmos necesarios para tratar las situaciones de fallo o reincorporación.
    - Se tiene la garantía de que todos perciben la misma secuencia de eventos de este tipo.

Índice

1. El problema de la pertenencia a grupo.
2. **Concepto de detector de fallos.**
3. Clasificación.
4. Aplicabilidad.

2. Concepto de detector de fallos

- Los detectores de fallos fueron definidos en [CT96] como una abstracción que...
  - Reside en cada uno de los procesos (o nodos) que componen el sistema distribuido.
  - Recoge la lista de procesos de los que se sospecha que hayan fallado.
  - Asume un modelo de fallos de caída (sin recuperación), con comunicación fiable y un sistema asincrónico.
- Nótese que las listas existentes en cada proceso-detector no tienen por qué coincidir.

2. Concepto de detector de fallos

- Para medir la calidad de un detector de fallos se utilizan dos propiedades:
  - *Completitud*: Los procesos que han fallado deben ser considerados sospechosos.
  - *Precisión*: Los procesos correctos no deben ser considerados sospechosos.
- Se distinguen dos grados de completitud y cuatro de precisión.

2. Concepto de detector de fallos

- Grados de completitud:
  - *Completitud fuerte*: Llega un momento en que todo proceso fallido es permanentemente sospechado por todo proceso correcto.
  - *Completitud débil*: Llega un momento en que todo proceso fallido es permanentemente sospechado por algún proceso correcto.

2. Concepto de detector de fallos

- Grados de precisión:
  - *Precisión fuerte*: Ningún proceso es sospechado antes de fallar.
  - *Precisión débil*: Existe algún proceso correcto del que jamás se sospecha.
  - *Precisión fuerte eventual*: Llega un momento en el que no se sospecha de ningún proceso correcto.
  - *Precisión débil eventual*: Llega un momento en el que no se sospecha de al menos un proceso correcto.

Índice

1. El problema de la pertenencia a grupo.
2. Concepto de detector de fallos.
3. **Clasificación.**
4. Aplicabilidad.

3. Clasificación

- Combinando los grados de completitud y precisión se generan 8 tipos de detectores:

Completeness	Accuracy			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\Diamond\mathcal{P}$	*Eventually Strong* $\Diamond\mathcal{S}$
Weak	$\mathcal{Q}$	*Weak* $\mathcal{W}$	$\Diamond\mathcal{Q}$	*Eventually Weak* $\Diamond\mathcal{W}$

FIG. 1. Eight classes of failure detectors defined in terms of accuracy and completeness.

3. Clasificación

- Pero [CT96] también proponen un algoritmo de reducción con el que se puede demostrar que a partir de un detector con completitud débil se puede implantar otro con completitud fuerte.
- *Reducción*: Un algoritmo $T_{D->D'}$ transforma un detector de clase D en otro de clase D' si y solo si para toda ejecución de $T_{D->D'}$ usando D, su salida pertenece al conjunto de salidas que es capaz de generar D'.

3. Clasificación

Every process p executes the following:

$output_p \leftarrow \emptyset$

cobegin
|| *Task 1:* **repeat forever**
{*p queries its local failure detector module* $\mathcal{D}_p$}
$suspects_p \leftarrow \mathcal{D}_p$
send $(p, suspects_p)$ to all

|| *Task 2:* **when** receive $(q, suspects_q)$ for some q
$output_p \leftarrow (output_p \cup suspects_q) - \{q\}$ {$output_p$ *emulates* $\mathcal{D}'_p$}
coend

FIG. 3. $T_{\mathcal{D}\rightarrow\mathcal{D}'}$: From Weak Completeness to Strong Completeness.

3. Clasificación

- El algoritmo anterior se basa en que los procesos que sospechen de la caída de algún otro difundirán sus sospechas, que serán aceptadas por todos los demás.
- Esto permite llegar a una completitud fuerte.
  - Así, bastaría con estudiar las cuatro clases de detectores con completitud fuerte.
  - Pero todo esto debe complementarse con cierto grado de precisión.
    - Se puede demostrar que la precisión es mantenida por este algoritmo de reducción.

Índice

1. El problema de la pertenencia a grupo.
2. Concepto de detector de fallos.
3. Clasificación.
4. **Aplicabilidad.**

4. Aplicabilidad

- Los autores de [CT96] utilizan los detectores para resolver dos problemas tradicionalmente irresolubles en sistemas distribuidos asíncronos donde puedan fallar los procesos:
  - Consenso.
    - Analizaremos este problema.
  - Difusión atómica.

4.1. Consenso

- El problema de consenso se define en base a dos operaciones (donde “v” representa un determinado valor perteneciente a un conjunto del que debe elegirse un elemento):
  - *proponer(v)*.
  - *decidir(v)*.
- Cada proceso puede ejecutar dichas operaciones. Se obtendrá una solución cuando todos los procesos decidan lo mismo.

4.1. Consenso

- Una solución correcta debe cumplir estas cuatro propiedades:
  - *Terminación*. Todo proceso correcto eventualmente decide algún valor.
  - *Integridad uniforme*. Todo proceso decide como máximo una vez.
  - *Acuerdo*. Ningún par de procesos correctos decide de manera diferente.
  - *Validez uniforme*. Si un proceso decide *v*, entonces *v* fue propuesto por algún proceso.

4.1. Consenso

- El problema de consenso puede resolverse con cualquier tipo de detector de fallos, pero...
  - Cada tipo soporta un número máximo de fallos diferente.
  - Cada tipo necesita un número de rondas de difusión fiable diferente.

4.1. Consenso

- Así...
  - Con la clase S (“strong”) se toleran n-1 fallos (siendo “n” el número de procesos que componen el sistema).
  - Con la clase diamante-S (“eventual strong”) se necesita una mayoría de procesos correctos (se toleran (n-1)/2 fallos).

4.1. Consenso

Every process p executes the following:

procedure *propose*(v_p)
  $V_p \leftarrow (\perp, \perp, \ldots, \perp)$ {*p's estimate of the proposed values*}
  $V_p[p] \leftarrow v_p$
  $\Delta_p \leftarrow V_p$

  Phase 1: {*asynchronous rounds* r_p, $1 \leq r_p \leq n-1$}
    for $r_p \leftarrow 1$ to $n-1$
      send (r_p, Δ_p, p) to all
      wait until $[\forall q$: received (r_p, Δ_q, q) **or** $q \in \mathcal{D}_p]$ {*query the failure detector*}
      $msgs_p[r_p] \leftarrow \{(r_p, \Delta_q, q) \mid$ received $(r_p, \Delta_q, q)\}$
      $\Delta_p \leftarrow (\perp, \perp, \ldots, \perp)$
      for $k \leftarrow 1$ to n
        if $V_p[k] = \perp$ **and** $\exists (r_p, \Delta_q, q) \in msgs_p[r_p]$ with $\Delta_q[k] \neq \perp$ **then**
          $V_p[k] \leftarrow \Delta_q[k]$
          $\Delta_p[k] \leftarrow \Delta_q[k]$

  Phase 2: send V_p to all
    wait until $[\forall q$: received V_q **or** $q \in \mathcal{D}_p]$ {*query the failure detector*}
    $lastmsgs_p \leftarrow \{V_q \mid$ received $V_q\}$
    for $k \leftarrow 1$ to n
      if $\exists V_q \in lastmsgs_p$ with $V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$

  Phase 3: *decide*(first non-$\perp$ component of V_p)

FIG. 5. Solving Consensus using any $\mathcal{D} \in \mathcal{S}$.

4.1. Consenso

- En el algoritmo anterior...
  - Al final de la fase 1, se han ejecutado n-1 rondas donde se han recogido las propuestas de todos los procesos.
    - Los procesos han podido fallar. Nada garantiza que todos hayan recibido las mismas propuestas.
  - Al final de la fase 2, se “limpian” aquellas componentes correspondientes a procesos que han sido incapaces de difundir su propuesta a algún destinatario correcto.
    - Entonces, todos tendrán vectores idénticos.
  - En la fase 3, todos los procesos correctos deciden lo mismo.
- Obsérvense las diferencias con el algoritmo siguiente...

4.1. Consenso

Every process p executes the following:

procedure *propose*(v_p)
  $estimate_p \leftarrow v_p$ {$estimate_p$ *is p's estimate of the decision value*}
  $state_p \leftarrow undecided$
  $r_p \leftarrow 0$ {r_p *is p's current round number*}
  $ts_p \leftarrow 0$ {ts_p *is the last round in which p updated* $estimate_p$, *initially 0*}

  {*Rotate through coordinators until decision is reached*}

  while $state_p = undecided$
    $r_p \leftarrow r_p + 1$
    $c_p \leftarrow (r_p \bmod n) + 1$ {c_p *is the current coordinator*}

    Phase 1: {*All processes p send* $estimate_p$ *to the current coordinator*}
      send $(p, r_p, estimate_p, ts_p)$ to c_p

    Phase 2: {*The current coordinator gathers* $\lceil \frac{(n+1)}{2} \rceil$ *estimates and proposes a new estimate*}
      if $p = c_p$ **then**
        wait until [for $\lceil \frac{(n+1)}{2} \rceil$ processes q : received $(q, r_p, estimate_q, ts_q)$ from q]
        $msgs_p[r_p] \leftarrow \{(q, r_p, estimate_q, ts_q) \mid p$ received $(q, r_p, estimate_q, ts_q)$ from $q\}$
        $t \leftarrow$ largest ts_q such that $(q, r_p, estimate_q, ts_q) \in msgs_p[r_p]$
        $estimate_p \leftarrow$ select one $estimate_q$ such that $(q, r_p, estimate_q, t) \in msgs_p[r_p]$
        send $(p, r_p, estimate_p)$ to all

4.1. Consenso

Phase 3: {*All processes wait for the new estimate proposed by the current coordinator*}
wait until [received $(c_p, r_p, estimate_{c_p})$ from c_p **or** $c_p \in \mathcal{D}_p$] {*Query the failure detector*}
if [received $(c_p, r_p, estimate_{c_p})$ from c_p] **then** {*p received* $estimate_{c_p}$ *from* c_p}
$estimate_p \leftarrow estimate_{c_p}$
$ts_p \leftarrow r_p$
send (p, r_p, ack) to c_p
else send $(p, r_p, nack)$ to c_p {*p suspects that* c_p *crashed*}

Phase 4: { *The current coordinator waits for* $\lceil \frac{(n+1)}{2} \rceil$ *replies. If they indicate that* $\lceil \frac{(n+1)}{2} \rceil$ *processes adopted its estimate, the coordinator R-broadcasts a decide message* }
if $p = c_p$ **then**
wait until [for $\lceil \frac{(n+1)}{2} \rceil$ processes q : received (q, r_p, ack) **or** $(q, r_p, nack)$]
if [for $\lceil \frac{(n+1)}{2} \rceil$ processes q : received (q, r_p, ack)] **then**
R-broadcast$(p, r_p, estimate_p, decide)$

{*If p R-delivers a decide message, p decides accordingly*}

when *R-deliver*$(q, r_q, estimate_q, decide)$
if $state_p = undecided$ **then**
$decide(estimate_q)$
$state_p \leftarrow decided$

FIG. 6. Solving Consensus using any $\mathcal{D} \in \diamond\mathcal{S}$.

4.1. Consenso

- Sin entrar en detalle sobre este segundo algoritmo, nótese que...
  - Al pasar de precisión débil a precisión débil eventual, se han necesitado muchas más fases para tomar una decisión.
  - Se tolera también un menor número de fallos.