# Sistemas Distribuídos
# Aula 11: O Consenso, sua Impossibilidade e os Detectores de Falhas

**Prof. Elias P. Duarte Jr.**
**Universidade Federal do Paraná (UFPR)**
**Departamento de Informática**
**www.inf.ufpr.br/elias/sisdis**

# Sumário

- Vamos iniciar a aula definindo um problema central em sistemas distribuídos: o consenso
- Em seguida vamos falar sobre um resultado importante, a impossibilidade FLP
- Com base no falado até aqui, vamos explicar como surgiu o conceito de detectores de falhas
- Além de definir os detectores, vamos examinar duas propriedades importantes: completude e precisão
- Classificação dos detectores de falhas baseada em completude e precisão

# O Consenso

- Considere um sistema distribuído que consiste de um conjunto de processos
- Informalmente, quando iniciam a execução do consenso, os processos “propõem” valores iniciais
  - cada processo propõe um valor de um conjunto de valores já definido, o clássico é {0,1}
- Ao final do consenso: todos os processos “decidem” pelo mesmo valor, que é um daqueles que foram propostos inicialmente

# Propriedades que Definem o Consenso

- Acordo: todos os processos corretos decidem o mesmo valor
- Validade: se um processo correto decide um determinado valor, então este valor foi proposto por um processo do sistema
- Terminação: todo processo correto eventualmente decide um valor

# O Consenso

- Muitos enxergam o consenso como o problema mais importante de sistemas distribuídos
- Consenso → Acordo Distribuído
- Diversas aplicações, em particular na replicação distribuída (“replicação máquina-de-estado”)
- Diversos algoritmos distribuídos são baseados (utilizam) o consenso: difusão atômica, transações distribuídas, etc.

# A Impossibilidade

- Em 1985, Fisher, Lynch e Paterson provaram que é impossível garantir a correta execução do consenso em sistemas distribuídos assíncronos, sujeitos a falhas crash
- “Impossibilidade FLP”
- A prova explora justamente o fato de que em um sistema assíncrono é impossível distinguir um processo lento de um processo falho
- O processo lento pode ser confundido como falho e acaba tomando uma decisão diferente dos demais

# Impossibilidade FLP

- A impossibilidade FLP é um resultado decisivo em Sistemas Distribuídos
- O consenso é muito importante, os sistemas reais não são síncronos: um resultado prático
- Diversos trabalhos propõem as mais diversas formas de contornar a impossibilidade FLP
  - justamente neste contexto foram inicialmente investigados os modelos parcialmente síncronos

# “Unreliable Failure Detectors”

- Um dos trabalhos mais importantes que surgiram na onda de tentar “contornar” a Impossibilidade FLP é o que definiu os Detectores de Falhas Não Confiáveis

# "Unreliable Failure Detectors"

- Um dos trabalhos mais importantes que surgiram na onda de tentar "contornar" a Impossibilidade FLP é o que definiu os Detectores de Falhas Não Confiáveis
- Em 1996, Chandra e Toueg pensaram no seguinte:

  → se os processos executando o consenso tiverem acesso a informação sobre a falha de processos, o consenso se torna possível?

  - quem falhou? quem está correto?

# "Unreliable Failure Detectors"

- Os detectores de falhas são – por definição – não confiáveis
- Podem cometer erros
- Neste contexto um erro pode ser de 2 tipos:

  → um processo que na realidade está falho é informado como correto pelo detector

  → um processo que na realidade está correto é informado como falho pelo detector

# Detector de Falhas: Oráculo

- Os franceses trouxeram esta visão sobre os detectores
- Eles são oráculos: os processos perguntam aos “oráculos” informações sobre estado dos demais processos
- Inspiração: oráculo de Delfos, na Grécia

# O Oráculo de Delfos

# Dectores de Falhas Não Confiáveis

- Os detectores de falhas são implementados como módulos aos quais os processos do sistema têm acesso localmente
- A saída (clássica) de um detector de falhas é a lista de processos suspeitos de terem falhado
- Atenção para a nomenclatura!
- Detectores de falhas são para sistemas assíncronos: como distinguir um processo falho de um processo lento? Impossível!
- Assim, o detector informa a “suspeita de falha” de um processo, não a “falha” de um processo
- Estados: **correto** e **suspeito**

# Propriedades dos Detectores

- Voltando ao início → será que os detectores de falhas permitem o consenso?
- A resposta para esta pergunda deve levar em conta todas as dificuldades vistas → os detectores de falhas cometem erros!
- Assim vamos refazer a pergunta original: quais as propriedades que um detector de falhas deve apresentar para permitir o consenso?

# Completude & Precisão

- Chandra & Toueg propuseram 2 propriedades básicas dos Detectores de Falhas:

  → *Completeness* (Completude): o detector de falhas suspeita de processos que efetivamente falharam

  → *Accuracy* (Precisão): o detector de falhas não suspeita de processos que efetivamente estão corretos

# Completude & Precisão

- Uma destas propriedades é, na verdade, fácil de conseguir – enquanto a outra é difícil
- Qual é qual?

# Completude & Precisão

- Uma destas propriedades é, na verdade, fácil de conseguir – enquanto a outra é difícil
- Qual é qual?
- Como processos são monitorados e um processo que falha no modelo crash não comunica de forma alguma, a completude é fácil!
  - observe que mesmo assim há um tempo entre a falha e a detecção da mesma ;-)
  - neste intervalo de tempo o detector vai dizer que o processo falho está correto :-0

# Precisão É Difícil

- A precisão é difícil de garantir – impossível?
- Pode acontecer, mas pode não acontecer
- Quer dizer, pode ser que eu jamais suspeite que um processo correto está falho, mas não tem como garantir isso
  - mesmo no dia a dia: por exemplo, você usando um computador remoto dá um <ENTER> demora mais um pouco do que esperado, e você já imagina se falhou!

# Eventual Properties

- Lembra-se que mesmo que a completude seja fácil, há um tempo até a detecção acontecer?
- Na propriedade é possível incorporar este fato dizendo que é *eventual*:

  → todo processo que falha é *eventually* suspeitado pelo detector

# Propriedades Fracas & Fortes

- As duas propriedades Completude e Precisão são reclassificadas como fortes e fracas
- Completude Forte: *eventually*, todo processo que falha é suspeitado por todo processo correto
- Completude Fraca: *eventually*, todo processo que falha é suspeitado por pelo menos 1 processo correto
- Note que a completude fraca pode ser transformada em forte: o processo que suspeita informa os demais corretos

# Precisão Forte e Fraca

- Precisão Forte: nenhum processo correto é suspeitado antes de falhar
- Precisão Fraca: pelo menos um processo correto jamais é suspeitado antes de falhar
- Há versões *eventually:*
  - Precisão Forte *Eventual:* existe um tempo após o qual nenhum processo é suspeitado antes de falhar
  - Precisão Forte *Eventual:* existe um tempo após o qual pelo menos 1 processo correto não é suspeitado

# Diversas Classes de Detectores

| Precisão → Completude ↓ | Forte | Fraca | Forte Eventual | Fraca Eventual |
|---|---|---|---|---|
| Forte | Perfeito **P** | Strong **S** | Evnt. Perfect **◊P** | ◊S |
| Fraca | **Q** | Weak **W** | ◊Q | Event. Weak **◊W** |

# Afinal: Os Detectores Permitem o Consenso?

- A resposta pode parecer surpreendente: basta que o detector de falhas seja ◊W para garantir o consenso!
- Mas não há como garantir, pode “acontecer” na execução ;-)

# Implementação de Detectores

- Os detectores de falhas podem ser implementados em dois modelos:

1) *Pull*: parecido com os testes do diagnóstico, processos perguntam explicitamente aos outros os seus estados

2) *Push:* este é o modelo mais usado – cada processo manda periodicamente mensagens chamadas "*heartbeats*" para todos os demais

# Detector de Falhas: Algoritmo Básico

Algoritmo Detector de Falhas executado pelo processo i;

**Init:** Suspeitos ← S - {i};
for all j ∈ S - {i}: compute timeout$_j$(heartbeat$_j$);
send(heartbeat$_i$) to j;

**Upon heartbeat$_i$ interval expires:** for all j ∈ S - {i}:
send(heartbeat$_i$) to q;

**Upon receive(heartbeat$_j$):** Suspeitos ← Suspeitos - {j}
update timeout$_j$(heartbeat$_j$);

**Upon timeout$_j$ expires:** Suspeitos ← Suspeitos + {j}

**Upon FailureDetector invoked:** return Suspeitos;

# Revolucionaram Sistemas Distribuídos

- Os detectores de falhas revolucionaram os sistemas distribuídos
- Não apenas o consenso foi revisto para incorporar detecção de falhas
- Mas **virtualmente todos** os problemas de sistemas distribuídos foram revisitados e classificados de acordo com o detector que precisam
- Por exemplo: a exclusão mútua distribuída precisa de detector perfeito, etc...

# Referências:

## Os Detectores de Falhas:

T. D. Chandra, S. Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems,” *Journal of the ACM*, Vol. 43, No. 2, 1996.

## A Impossibilidade FLP:

M. J. Fischer, N. Lynch, M.S. Paterson, “Impossibility of Distributed Consensus with One Faulty Process,” *Journal of the ACM*, Vol. 32, No. 2, 1985.

**Todos são conceitos discutidos de formas diversas nos livros sugeridos.**

# Conclusão

- Nesta aula primeiro definimos de maneira informal o consenso
- Depois foi vista a impossibilidade FLP
- Os detectores de falhas foram então introduzidos
- E foram definidas as propriedades de completude e precisão
- Terminamos com a classificação de detectores de falhas
- Mesmo o detector mais fraco resolve o consenso, mas...

**Obrigado!**
**Página da Disciplina**
**Sistemas Distribuídos:**
**www.inf.ufpr.br/elias/sisdis**