ACH 2147 — Desenvolvimento de Sistemas de Informação Distribuídos

Aula 23: Tolerância a falhas (parte 1)

Prof. Renan Alves

Escola de Artes, Ciências e Humanidades — EACH — USP

27/05/2024

Dependabilidade

Ideia básica

Um componente provê serviços para clientes. Para prover estes serviços, o componente pode precisar de serviços de outros componentes $\Rightarrow$ neste caso, o componente depende de outro(s) componente(s)

Sendo mais específico:

Um componente C depende de outro componente $C*$ se a corretude do comportamento de C depender da corretude do comportamento de $C*$.

Obs.: componente = processo ou canal de comunicação

Dependabilidade

Ideia básica

Um componente provê serviços para clientes. Para prover estes serviços, o componente pode precisar de serviços de outros componentes $\Rightarrow$ neste caso, o componente depende de outro(s) componente(s)

Sendo mais específico:

Um componente C depende de outro componente $C*$ se a corretude do comportamento de C depender da corretude do comportamento de $C*$.

Obs.: componente = processo ou canal de comunicação

Requisitos relacionados a dependabilidade

Requisito	**Descrição**
Disponibilidade	Prontidão de uso
Confiabilidade	Continuidade do serviço
Segurança (Safety)	Baixas chances de catástrofes
Manutenibilidade	Facilidade de reparar uma falha

Confiabilidade versus Disponibilidade

Confiabilidade $R(t)$ de um componente C

Probabilidade condicional de C estar funcionando corretamente no intervalo $[0, t)$, dado que estava funcionando corretamente em $T = 0$.

Métricas tradicionais

- Mean Time To Failure (*MTTF*): tempo médio até que o componente falhe.
- Mean Time To Repair (*MTTR*): tempo médio de reparo.
- Mean Time Between Failures (*MTBF*): apenas $MTTF + MTTR$.

Confiabilidade versus Disponibilidade

Disponibilidade $A(t)$ de um componente C

Fração média do tempo em que C esteve funcionando no intervalo $[0, t)$.

- Disponibilidade de longo prazo A: $A(\infty)$
- Note: $A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF+MTTR}$

Observação

Confiabilidade e disponibilidade fazem sentido apenas se tivermos uma noção precisa do que uma falha é de fato.

Terminologia

Falha, erro, defeito

Termo	Descrição	Exemplo
Falha (failure)	Um componente não está operando de acordo com a especificação	Programa deu crash
Erro	Parte do componente que pode levar a falhas	Bug na implementação
Defeito (fault)	Causa do erro	Programador desatento

Terminologia

Formas de lidar com defeitos

Termo	Descrição	Exemplo
Prevenção de defeitos	Evita a existência de defeito	Não contratar programadores desatentos
Tolerância a defeitos	Componente capaz de mascarar a ocorrência de um defeito	Construir mais de uma versão do componente, de forma independente
Remoção de defeitos	Reduzir a quantidade ou seriedade dos defeitos	Demitir os programadores desatentos
Previsão de defeitos	Estimar a presença atual, incidência futura e as consequência dos defeitos	Estimar a quantidade de programadores desatentos contratados

Modelos de falhas

Tipos de falhas

Tipo	Descrição do comportamento do servidor
Falha de parada (crash)	Para de funcionar, mas funcionava corretamente até parar
Falha de omissão	Deixa de responder requisições
Omissão de RX	Falha em receber mensagens
Omissão de TX	Falha em enviar mensagens
Falha temporal	Resposta fora do prazo estabelecido
Falha de resposta	Resposta incorreta
Falha de valor	O calor da resposta está incorreto
Falha de estado	Desvio do fluxo de controle correto
Falha arbitrária	Possibilidade de gerar respostas arbitrárias em intervalos arbitrários, indetectável se a resposta está correta ou não

Dependabilidade versus segurança

Omissão versus ação

Falhas arbitrárias podem ser maliciosas. Pode-se fazer a seguinte distinção:

- Falha de omissão: um componente falha em realizar uma ação que deveria ter realizado
- Falha de ação: um componente realiza uma ação que não ter realizado

Dependabilidade versus segurança

Omissão versus ação

Falhas arbitrárias podem ser maliciosas. Pode-se fazer a seguinte distinção:

- Falha de omissão: um componente falha em realizar uma ação que deveria ter realizado
- Falha de ação: um componente realiza uma ação que não ter realizado

Observação

Note que falhas deliberadas, sejam elas de omissão ou de ação, são tipicamente problemas de segurança (security). Distinguir entre falhas deliberadas e falhas não intencionais é impossível, no caso geral.

Falhas de parada

Cenário

C não percebe mais nenhuma atividade de C^* — seria uma falha de parada? Distinguir entre um crash e uma falha de omissão ou falha temporal pode ser impossível.

Sistneas assíncronos vs síncronos

- Sistemas assíncronos: nada pode ser assumido a respeito da velocidade de execução dos processos ou sobre quando as mensagens serão entregues $\rightarrow$ não é possível detectar falhas de parada de forma confiável.
- Sistemas síncronos: a velocidade de execução dos processos e o momento de entrega das mensagens são delimitados $\rightarrow$ é possível detectar falhas de parada de forma confiável.
- Na práticas, os sistemas são parcialmente síncronos: na maior parte do tempo podemos assumir que o sistema é síncrono, porém os limites de tempo podem ser ocasionalmente violados $\rightarrow$ na maioria das vezes é possível detectar crashes corretamente.

Falhas de parada

Classificações

Tipo de parada	Descrição
Fail-stop	Falhas de parada, detectáveis de forma confiável
Fail-noisy	Falhas de parada, eventualmente detectáveis de forma confiável
Fail-silent	Falhas de omissão ou crash: clientes não conseguem distinguir
Fail-safe	Falha arbitrária, porém benigna (i.e., não causarão dano)
Fail-arbitrary	Falha arbitrária e maliciosa

Mascarando falhas através de redundância

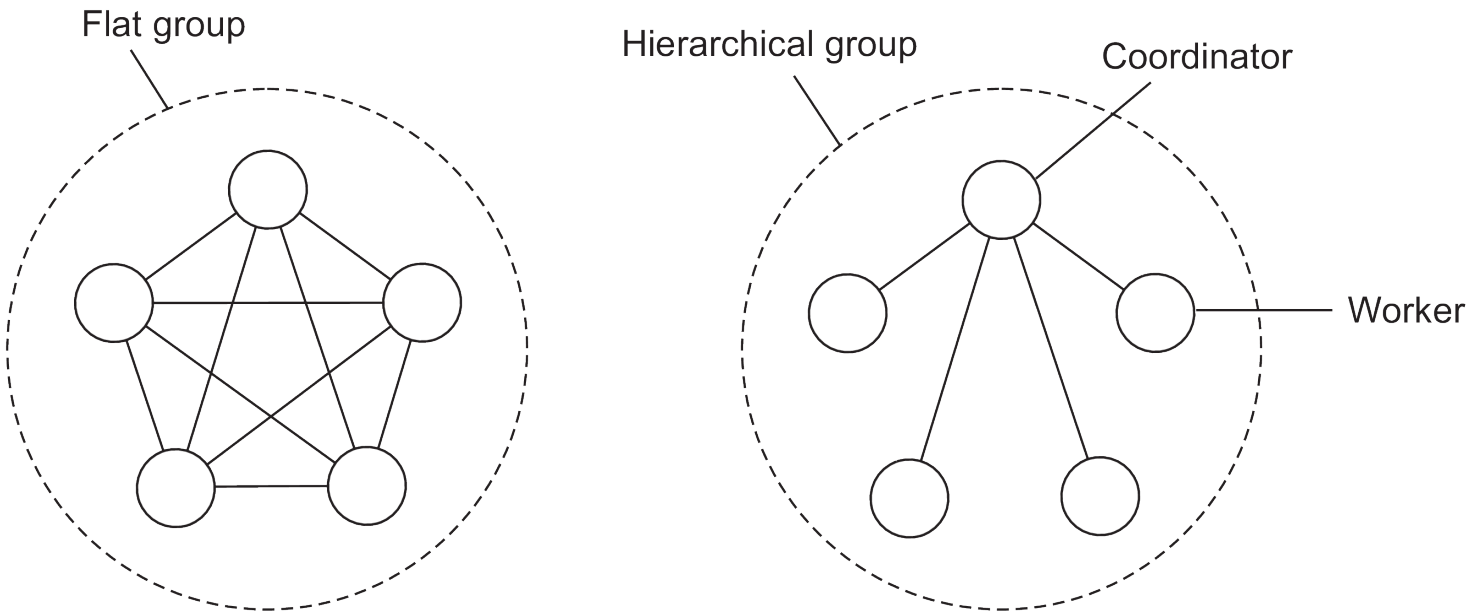
Tipos de redundância

- Redundância de informação: Adição de bits à unidades de dados de forma que o dado original pode ser recuperado mesmo que alguns bits sejam corrompidos
- Redundância de tempo: Sistema projetado de tal forma que ações podem ser executadas novamente em caso de falhas. Tipicamente usado para falhas transientes/intermitentes.
- Redundância física: Uso de equipamentos e/ou processos adicionais, de forma que alguns deles possam falhar sem afetar o sistema como um todo. Tipo de redundância mais comum em sistemas distribuídos.

Resiliência de processos

Ideia básica

Proteger o sistema contra falhas de processos através da replicação de processos, organizando-os em grupos de processos.
Os grupos podem ser planos ou hierárquicos.



É preciso gerenciar a entrada e saída de processos do grupo.

Grupos e mascaramento de falhas

Grupo com grau de tolerância k

Grupo capaz de mascarar até k processos falhando ao mesmo tempo

Grupos e mascaramento de falhas

Grupo com grau de tolerância k

Grupo capaz de mascarar até k processos falhando ao mesmo tempo

Qual deve ser o tamanho do grupo para ter grau de tolerância k?

- Considerando falhas de parada ou omissão: são necessários $k+1$ membros no grupo. Nenhum membro produzirá resultado incorreto então basta termos ao menos uma resposta.
- Considerando falhas arbitrárias: são necessários $2k+1$ membros. Resultado correto obtido através de votação da maioria

Grupos e mascaramento de falhas

Grupo com grau de tolerância k

Grupo capaz de mascarar até k processos falhando ao mesmo tempo

Qual deve ser o tamanho do grupo para ter grau de tolerância k?

- Considerando falhas de parada ou omissão: são necessários $k+1$ membros no grupo. Nenhum membro produzirá resultado incorreto então basta termos ao menos uma resposta.
- Considerando falhas arbitrárias: são necessários $2k+1$ membros. Resultado correto obtido através de votação da maioria

Hipóteses

- Todos os membros são idênticos
- Todos os membros processam comandos na mesma ordem

Resultado: podemos afirmar que todos os processos executam exatamente a mesma coisa.

Consenso

Pré-requisito

Dentro do grupo, cada processo que está funcionando (não está em falha) executa os mesmos comando e na mesma ordem.

Reformulação

Deve haver consenso entre os membros que não estão em falha a respeito de qual comando será o próximo a ser executado.

Consenso baseado em flooding

Modelo do sistema

- Um grupo de processos $\mathbf{P} = \{P_1, \ldots, P_n\}$
- Falhas fail-stop, i.e., com detecção confiável de falhas
- Um cliente contacta um processo P_i requisitando a execução de um comando
- Cada processo P_i mantém uma lista de comandos propostos

Consenso baseado em flooding

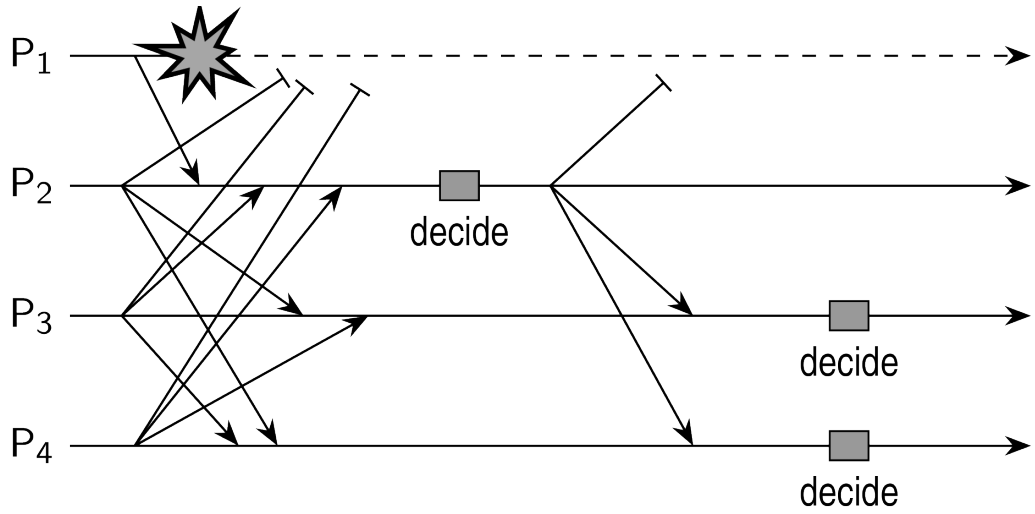
Modelo do sistema

- Um grupo de processos $\mathbf{P} = \{P_1, \ldots, P_n\}$
- Falhas fail-stop, i.e., com detecção confiável de falhas
- Um cliente contacta um processo P_i requisitando a execução de um comando
- Cada processo P_i mantém uma lista de comandos propostos

Algoritmo básico (em rodadas)

1. Na rodada r, P_i envia por multicast o seu conjunto de comandos conhecido $\mathbf{C_i^r}$ para todos os outros
2. No final da rodada r, cada P_i concatena todos os comandos recebidos em um novo conjunto $\mathbf{C_i^{r+1}}$.
3. O próximo comando cmd_i é selecionado através de uma função globalmente compartilhada e determinística: $cmd_i \leftarrow select(\mathbf{C_i^{r+1}})$.

Consenso baseado em flooding: exemplo



Observações

- P_2 recebeu todos os comandos propostos por todos os outros processos $\Rightarrow$ faz uma decisão.
- P_3 pode ter detectado que P_1 crashou, mas não sabe se P_2 recebeu alguma coisa, i.e., P_3 não sabe se ele tem a mesma informação que P_2 $\Rightarrow$ não pode fazer uma decisão (idem para P_4).

Protocolo Raft

Desenvolvido para compreensibilidade

- Usa um mecanismo simples de eleição de líder. O líder atual opera durante o term atual.
- Cada servidor (geralmente 5) mantém um log das operações, algumas das quais já foram efetuadas. Um servidor de backup não vota em novo líder se o seu próprio log for mais recente.
- Todas as operações efeutadas tem a mesma posição no log de cada um dos servidores.
- O líder decide qual operação pendente será a próxima a ser efetuada $\Rightarrow$ uma abordagem baseada em primário.

Raft

Processando novas requisições

- Um cliente envia uma requisição para efeutar a operação o.
- O líder adiciona a requisição $\langle o, t, k \rangle$ no seu log (registrando o term atual t e índice k).
- O log é enviado para todos os outros servidores
- Os outros servidores copiam o log e confirmam o recebimento.
- Quando a maioria dos servidores confirmarem, o líder efetua a operação o e a resposta pode ser enviada ao cliente.

Raft

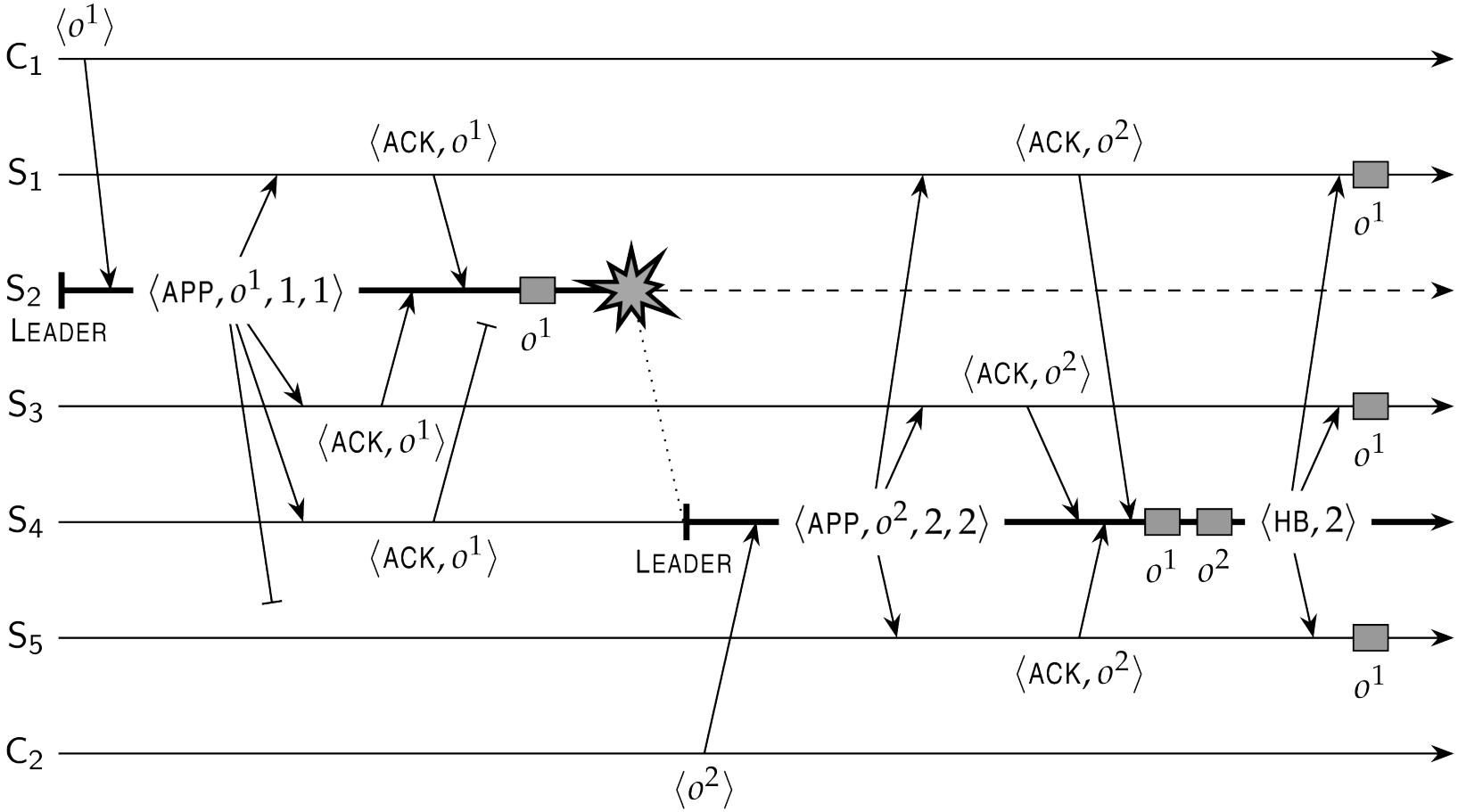
Processando novas requisições

- Um cliente envia uma requisição para efeutar a operação o.
- O líder adiciona a requisição $\langle o, t, k \rangle$ no seu log (registrando o term atual t e índice k).
- O log é enviado para todos os outros servidores
- Os outros servidores copiam o log e confirmam o recebimento.
- Quando a maioria dos servidores confirmarem, o líder efetua a operação o e a resposta pode ser enviada ao cliente.

Nota

Na prática, apenas atualizações são transmitidas. No fim, cada servidor tem a mesma visão e sabe sobre as operações efetuadas. As informações nos servidores (logs) de backup são sobrescritas pelas informações do líder se necessário.

Raft: falha do líder



Observações

O novo líder será aquele que tiver o maior número de operações efetuadas em seu log. Os outros servidores de backup receberão uma cópia da versão mais recente eventualmente.