Blocos Básicos em Sistemas Distribuídos Tolerante a Falhas

O objetivo de sistemas de alta disponibilidade (ou seja, tolerante a falhas) é continuar a prover serviços mesmo na ocorrência de falhas em alguns de seus componentes. Existem diferentes métodos e esquemas para promover tal capacidade a sistemas distribuídos. Muitos deles fazem suposições implícitas ou explícitas sobre o comportamento do sistema e de seus componentes, e sobre seus modos de falha. Como estas suposições não são satisfeitas na prática, devem existir mecanismos de poder atingi-las, a fim de que estes esquemas possam funcionar. Desta maneira, estas suposições sobre o comportamento de sistemas distribuídos podem ser vistas como vários blocos básicos usados na construção de um sistema tolerante a falhas. Este seção descreverá cada um destes blocos básicos, e explorará os possíveis métodos de se atingir os objetivos.

Acordo Bizantino (*Byzantine Agreement*)

Supõe-se freqüentemente que quando um componente falha, ele se comporta de uma maneira bem definida, apesar deste comportamento ser diferente do comportamento normal esperado. Entretanto, normalmente quando um componente falha ele se comporta de maneira totalmente arbitrária. O problema de alcançar um entendimento entre os componentes de um sistema distribuído, onde eles podem falhar de maneira arbitrária, é chamado de *Problemas Gerais Bizantinos*. Os protocolos usados para se atingir este entendimento são chamados de *Protocolos de Acordo Bizantino*.

Definição do Problema: seja um sistema distribuído com vários nós, onde eles trocam informações entre si através de mensagens. Os nós podem falhar, e um nó defeituoso pode mandar valores diferentes a nós distintos, relativos à mesma informação. O objetivo básico é atingir um consenso entre os nós não defeituosos sobre os valore corretos. Cada nó deve tomar uma decisão baseada nos valores recebidos dos outros nós, e todos os nós não defeituosos devem tomar a mesma decisão.

Relógios Sincronizados (*Synchronized Clocks*)

- Um sistema distribuído consiste em um grupo de máquinas independentes fracamente agrupadas. Desta maneira, cada uma possui seu relógio interno. Como relógios são processos físicos, eles naturalmente diferem um dos outros. Através de relógios lógicos é possível definir uma ordem total e parcial de eventos, apesar de não possuírem nenhuma relação com uma ordenação baseada em tempo real. Desta maneira, relógios sincronizados é um requisito para algumas aplicações distribuídas. Ainda, a diferença de relógios em um sistema distribuído pode levar a sérios problemas de coordenação do cluster.

Definição do Problema Seja um cluster onde há um relógio físico em cada nó, o qual é controlado por um processo neste nó. Como este processo que o controla é quem informa a hora marcada no relógio, uma falha no relógio pode ser vista como uma falha neste processo. Para a sincronização de relógios, é considerado um modelo que pode captar tanto uma falha no processo como uma falha no relógio.

Repositório Estável (*Stable Storage*)

Técnicas de tolerância a falhas freqüentemente necessitam que algum estado do sistema esteja disponível após uma falha. Ou seja, supõe-se que o sistema possui algum *repositório estável*, cujo conteúdo é preservado na sua integridade mesmo em caso de falhas. Devido a suas características, um disco comum não é aceitável como um repositório confiável. Nesta seção

será definido precisamente o que é um repositório estável, e serão descritas algumas técnicas para construí-lo.

Definição do Problema; basicamente, o problema resume-se em tornar estável para aplicações um sistema de armazenamento comum, com seus vários modos de falha. Assim como nenhum sistema pode ser tolerante a todos os possíveis tipos de falhas, um sistema estável de armazenamento aumenta sua confiabilidade tornando-se tolerante aos tipos de falhas mais prováveis e comuns. Para entender o problema é necessário primeiro estudar o sistema físico sobre o qual o repositório será construído, assim como seus modos de falha. Tipicamente, um repositório estável é construído a partir de um sistema de discos. Um sistema de armazenamento físico de discos é modelado como um conjunto de páginas (ou blocos ou setores) que possuem blocos de informação e um status associado a cada bloco, que pode ser *bom* ou *ruim*, dependendo se a informação está correta ou corrompida. Para interagir com o disco, processos podem usar duas operações: **procedure** read(adr) returns (status,data) e **procedure** write(addr, data). A primeira lê um endereço do disco, retornando seu status e informação, e a segunda grava uma informação em uma determinada posição no disco.

Falhas em um Sistema de Armazenamento baseado em Discos: vários tipos de erros podem surgir em sistemas de discos. Muitos deles podem ser tratados com técnicas de codificação, outros não, como por exemplo:

1. *Falhas Transientes*. Estas fazem com que o disco se comporte imprevisivelmente por um período de curta duração.
2. *Setor Ruim* (*Bad Sector*). Uma página torna-se corrompida, e a informação nela armazanada não pode ser lida. Isto poderia ocorrer devido a razões físicas.
3. *Falha da Controladora*. A controladora do disco falha, tornando o conteúdo do disco inacessível, porém não corrompido.
4. *Falha do Disco*. O disco todo se torna ilegível. Este tipo de falha poderia ocorrer devido a uma falha de hardware, e o disco não pode mais ser recuperado.

1. Ao invés de enumerar todas as possíveis falhas físicas em um disco, é melhor estudar suas manifestações no comportamento do disco e então prover métodos de tolerar estes erros. Os eventos que ocorrem em um sistema de discos compreendem os eventos gerados por operações de read/write e alguns eventos espontâneos. Alguns destes eventos são desejados (como um read retornando a informação correta), enquanto outros são indesejados, sendo estes causados por falhas físicas no disco. Para construir um sistema de armazenamento estável, o objetivo deve ser mascarar os eventos indesejados.

Processadores Fail Stop (*Fail Stop Processors*)

Geralmente, os esquemas de tolerância a falhas em um cluster fazem algumas suposições sobre o comportamento de nós defeituosos. Freqüentemente, supõe-se que, em caso de falha, o nó não irá executar operações inválidas e simplesmente deixará de funcionar. Nós que apresentam este tipo de comportamento são chamados de *processadores fail stop* (*fail stop processors*). Nesta seção será explicado em detalhes tal conceito, além de como esta abstração pode ser implementada por processadores normais.

Definição do Problema Um processador possui um conjunto de instruções, que são aquelas que ele pode executar. A execução de uma instrução pode alterar o estado interno do processador e também o estado da memória que o processador acessa. Para um processador que funciona perfeitamente, os efeitos de se executar diferentes instruções pode ser precisamente definido, enquanto que em um processador defeituoso, seu comportamento pode ser arbitrário. Geralmente, quando um processador falha, nada pode ser afirmado sobre seu comportamento, exceto que é diferente do comportamento previsto. Tal modo de falha genérico tornará a tarefa de se construir um sistema tolerante a falhas extremamente complexa, devido às inúmeras possibilidades de comportamentos durante uma falha. Um *processador fail stop* é caracterizado por seu modo de falha extremamente simples. Durante uma falha, o processador simplesmente pára de funcionar. Ainda, o estado interno do processador e o conteúdo da memória volátil à

qual ele está anexado são perdidos para sempre; o conteúdo do repositório estável anexado ao processador não é afetado por tal falha. Além disso, supõe-se geralmente que a falha de um processador fail stop pode ser detectada por outros processadores. Assim, os efeitos visuais de uma falha de um processador fail stop são:

1. Ele pára de executar.
2. O estado interno e o conteúdo da memória volátil conectada ao processador são perdidos; o conteúdo do repositório estável é inalterado.
3. Qualquer outro processador pode detectar a falha de um processador fail stop.

Detecção de Avarias e Diagnóstico de Falhas (*Failure Detection and Fault Diagnosis*)

Uma vez que algum componente em um sistema distribuído falha, o objetivo de um sistema tolerante a falhas é mascarar esta falha para clientes externos. Isto significa que outros componentes devem executar atividades extras além das suas normais, no caso uma falha de algum componente. Assim, a falha de um componente deve ser detectada e diagnosticada por outros componentes. A maioria das aplicações e métodos tolerantes a falhas em sistemas distribuídos assumem que uma vez que um nó falha, outros nós perceberão esta falha dentro de um período de duração finita, e então executarão as medidas de reparo necessárias. Este requisito é facilmente satisfeito por um método de força bruta, onde cada nó utiliza timeouts para detectar falhas. Entretanto, em um grande sistema, esta abordagem pode ser ter um custo alto demais. Nesta seção será discutido o tema de diagnóstico de falhas em sistemas distribuídos, no qual cada componente não defeituoso detecta a falha de outros componentes.

Entrega Confiável de Mensagens (*Reliable Message Delivery*)

Em sistemas distribuídos, supõe-se freqüentemente que uma mensagem enviada por um nó chega incorrupta ao seu destino, e que a ordem das mensagens é preservada entre dois nós. Isto é, se um nó (1) envia várias mensagens para outro nó (2), então o nó (2) recebe as mensagens na mesma ordem em que foram enviadas. Entretanto, perdas de mensagens e erros de comunicação são fatos em meios de comunicação reais. Assim, são necessários mecanismos para satisfazer estes requisitos. Na maioria dos esquemas de tolerância a falhas em sistemas distribuídos supõe-se implicitamente que estas duas propriedades são satisfeitas mesmo em caso de falhas de nós, dado que os dois nós envolvidos na comunicação permaneçam conectados. Enquanto que comunicação ordenada e confiável é uma propriedade genérica, estes requisitos adicionais são particularmente interessantes para tolerância a falhas.

Broadcast de Mensagens

Apesar de comunicação ponto-a-ponto ser suficiente para muitas aplicações, há muitas aplicações que necessitam que um nó envie uma mensagem para vários nós. *Broadcast* é a forma de comunicação onde um nó envia uma mensagem a todos os outros nós do grupo. Como a primitiva básica de comunicação suportada por uma rede é ponto-a-ponto, esta primitiva deve ser usada para implementar broadcast. Desta forma, a implementação está suscetível a falhas de nós e de comunicação, e isto não é aceitável em sistemas tolerantes a falhas. Basicamente, há três propriedades que interessam quando mensagens são enviadas a diferentes nós:

- *confiabilidade*: a mensagem deve ser recebida por todos os nós em operação.
- *ordenação consistente*: mensagens diferentes enviadas por nós diferentes devem ser entregues a todos os nós na mesma ordem.
- *preservação da casualidade*: a ordem na qual as mensagens são enviadas deve ser consistente com a casualidade entre os eventos de envio destas mensagens.