MC714

Sistemas Distribuídos

1° semestre, 2017

Primitivas para comunicação distribuída

Síncrona / assíncrona

- Primitiva síncrona
  - Primitivas send() e receive() são síncronas se fazem *handshake.*
  - Processamento do *send* só termina depois que *receive* correspondente foi invocado e a operação de recebimento foi terminada.
  - Receive termina somente quando dados copiados ao buffer de recepção do usuário.
- Primitiva assíncrona
  - Primitiva *send* é assíncrona se controle retorna ao processo depois dos dados serem copiados para fora do buffer do usuário.
  - Não há sentido em definir uma primitiva receive assíncrona.

Bloqueante / não bloqueante

- Primitiva bloqueante
  - Controle retorna ao processo após o processamento da primitiva (síncrona ou assíncrona) ser completado.
- Primitiva não bloqueante
  - Controle retorna ao processo imediatamente após invocá-la, mesmo com a operação não terminada.
  - Para *send*, controle retorna ao processo antes da cópia dos dados para fora do buffer do usuário.
  - Para *receive*, controle pode retornar ao processo mesmo antes dos dados chegarem do remetente.

Bloqueante / não bloqueante

- Quatro versões da primitiva send:
  - *Síncrona bloqueante*
  - *Síncrona não bloqueante*
  - *Assíncrona bloqueante*
  - *Assíncrona não bloqueante*
- Duas versões da primitiva receive:
  - Síncrona bloqueante
  - Síncrona não bloqueante

Emulação

- Vimos que memória compartilhada pode ser emulada em um sistema de troca de mensagens e vice-versa.
- Quatro classes de programas.
  - Assíncrono, troca de mensagens (AMP)
  - Síncrono, troca de mensagens (SMP)
  - Assíncrono, memória compartilhada (ASM)
  - Síncrono, memória compartilhada (SSM)
- Se um sistema A pode ser emulado por um sistema B, e se um problema não pode ser resolvido em B, também não pode ser em A.
- Se pode ser resolvido em A, pode ser resolvido em B.
- Fig 18

Emulação

- As quatro classes oferecem “computabilidade” (i.e., que problemas podem e não podem ser resolvidos) equivalente em sistemas livres de falhas.
- Em sistemas suscetíveis a falhas, sistemas síncronos oferecem maior “computabilidade” que assíncronos.

Desafios

Desafios

- Primórdios - 1970/ARPANET
  - Acesso a dados remotos sob falhas
  - Projeto de sistemas de arquivos
  - Projeto de estrutura de diretórios
  - Outros requisitos e problemas surgiram com o passar do tempo

- Relacionados a sistemas
- Relacionados a algoritmos
- Relacionados a tecnologias/aplicações recentes

Desafios - Sistema

- Comunicação: mecanismos apropriados para comunicação entre processos.
- Processos: gerência de processos e threads em clientes/servidores; migração de código; software e agentes móveis.
- Nomenclatura: esquemas de nomenclatura robustos para localização de recursos e processos transparentemente, em especial para sistemas móveis.
- Sincronização: coordenação entre processos é essencial. Outras formas de exclusão mútua, sincronização de relógio, relógios lógicos, algoritmos para manutenção global de estado.

Desafios - Sistema

- Armazenamento e acesso a dados: esquemas de armazenamento, acesso rápido e escalável a dados na rede. Projeto de sistemas de arquivo direcionados a sistemas distribuídos.
- Consistência e replicação: evitar gargalos; fornecer acesso rápido a dados e escalabilidade. Gerência de réplicas e consistência.
- Tolerância a falhas: manter funcionamento eficiente mesmo na presença de falhas de enlace, processos ou nós. Resiliência de processos, comunicação confiável, checkpointing e recuperação, detecção de falhas.
- Segurança

Desafios - Sistema

- Transparência em diversos níveis: acesso, localização, migração, relocação, replicação, concorrência, falha.
- Escalabilidade e modularidade: algoritmos, dados e serviços tanto distribuídos quanto possível. Técnicas como replicação, caching e gerência de cache, e processamento assíncrono ajudam a alcançar escalabilidade.

Desafios – Algorítmicos

- Modelos de execução e arcabouços: especificações formais de modelos e conjuntos de ferramentas para raciocínio, projeto e análise de programas distribuídos.
- Algoritmos em grafos dinâmicos: algoritmos importantes para comunicação, disseminação de dados, localização de objetos, busca de objetos. Topologia e conteúdo dinâmicos, algoritmos influenciam latência, tráfego e congestionamento.

Desafios – Algorítmicos

- Comunicação em grupo, multicast, entrega de mensagens ordenadas: algoritmos para comunicação e gerência eficiente de grupos dinâmicos sob falhas.
- Monitoramento: algoritmos online para monitorar e mapear eventos e tomar ações.
- Ferramentas para desenvolvimento e teste de programas distribuídos.
- Depuração de programas distribuídos: desafio devido à concorrência; mecanismos e ferramentas são necessários.
- Algoritmos para gerência e replicação de dados, atualização de réplicas e cópias em cache. Alocação de cópias no sistema.

Desafios – Algorítmicos

- WWW – cache, busca, escalonamento: prefetching/padrões de acesso, redução de latência de acesso, busca de objetos, distribuição de carga.
- Abstrações de memória compartilhada.
- Confiabilidade e tolerância a falhas.
- Balanceamento de carga: aumentar vazão e diminuir latência. Migração de dados, migração de computação, escalonamento distribuído.
- Escalonamento em tempo real: aplicações sensíveis ao tempo. Problema se visão global do sistema não está disponível. Difícil prever atrasos de mensagens.

Desafios – Algorítmicos

- Desempenho: alta vazão pode não ser o objetivo primário de um sistema distribuído, mas desempenho é importante.
  - Métricas: identificação de métricas apropriadas para medir desempenho teórico e prático (medidas de complexidade versus medida com parâmetros/estatísticas reais).
  - Ferramentas e métodos de medição: metodologias apropriadas e precisas para obtenção de dados confiáveis para as métricas definidas.

Desafios - Aplicações e avanços recentes

- Sistemas móveis: meio de broadcast compartilhado; problemas de alcance e interferência. Roteamento, gerência de localidade, alocação de canais, estimativa de posição, são problemas a serem tratados. Estação-base versus ad-hoc.
- Redes sensores: monitorar eventos distribuídos / streaming de eventos. Economia de energia na comunicação/middleware.
- Computação ubíqua e internet das coisas: auto-organização, recursos limitados; inteligência no núcleo da rede?
- Peer-to-peer: mecanismos de armazenamento persistente, busca e obtenção escalável, privacidade.

Desafios - Aplicações e avanços recentes

- Publish-subscribe, distribuição de conteúdo e multimídia: filtros de informação dinâmicos. Mecanismos eficientes para distribuição aos interessados e de inscrição de interessados. Mecanismos de agregação de informação e de distribuição de dados com grande volume.
- Grades computacionais: ciclos de CPUs ociosas; escalonamento e garantias de tempo real; predição de desempenho; segurança.
- Nuvens computacionais: virtualização; escalonamento / custos monetários; modelos econômicos; segurança.
- Segurança: soluções escaláveis para confidencialidade, autenticação e disponibilidade sob ações maliciosas.

Fundamentos de Sistemas Distribuídos

Histórico

- 1945 – 1985
  - computadores grandes e caros.
  - Independentes (falta de meios para conectá-los).
- ~1985 em diante, dois avanços:
  - Microprocessadores com maior “velocidade”.
    - Capacidade de processamento de um mainframe, mas uma fração do preço.
  - Redes de alta velocidade
    - Redes locais (LANs): centenas de máquinas conectadas com troca “rápida” de informações.
    - Redes de longa distância (WANs): milhões de máquinas espalhadas geograficamente podem comunicar-se.
- Mudança de paradigma: sistemas centralizados → sistemas distribuídos

Sistema Distribuído

- Um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente.
  - Consiste de componentes = computadores / dispositivos.
  - Usuários (pessoas ou programas): acham que é um único sistema centralizado.
  - Componentes autônomos precisam colaborar.
  - Como estabelecer tal colaboração = cerne do desenvolvimento de sistemas distribuídos.
  - Não há restrição de tipos de componentes ou como tais componentes se conectam: podem até estar dentro de um único sistema (como vimos nos sistemas multiprocessados...).

Sistema Distribuído

- Modo como se comunicam: oculto do usuário.
- Organização interna: oculta do usuário.
- Usuários / aplicações: podem interagir com o SD de maneira consistente e uniforme, independente de localização.
- Fácil de expandir / escalável.
- Continuamente disponível (como um todo – partes podem falhar ou deixar o sistema independentemente).

Middleware

- Suporte a computadores e redes heterogêneas: camada de software.
- Entre aplicações/usuário e S.O.
- Middleware se estende por múltiplas máquinas e oferece a mesma interface para diversas aplicações.
- Proporciona meios para que componentes de uma aplicação distribuída se comuniquem.
- Oculta diferenças de hardware e sistemas operacionais.
- Fig 19.

Metas

- Fácil acesso a recursos;
- Ocultar distribuição dos recursos;
- Ser aberto para poder ser expandido (escalabilidade).

Metas – Acesso aos recursos

- Facilitar acesso a recursos remotos e seu compartilhamento de maneira controlada e eficiente.
  - Recursos incluem hardware, software, dados.
- Uma razão óbvia: economia.
  - 1 impressora para todos
  - 1 supercomputador para todos
  - 1 sistema de armazenamento de alto desempenho para todos.
  - ...
- Colaboração
  - Edição online de documentos, imagens, apresentações, etc. em tempo real, comércio eletrônico

Metas – Acesso aos recursos

- Segurança: hoje não há muita proteção (cartões de crédito, privacidade, senhas em forma de texto, etc).
- Rastreamento de informações / violação de privacidade.
  - Montagem de perfil para propaganda/spam
  - Uso de filtros

Metas - Transparência

- Ocultar o fato de recursos estarem distribuídos.
- Sistema distribuído que apresenta-se a usuários e aplicações como se fosse um sistema único é denominado **transparente.**
- Diferentes aspectos de transparência / necessidade de transparência.

Metas - Transparência

- Ocultar o fato de recursos estarem distribuídos.
- Diferentes aspectos de transparência.
- Acesso: esconde diferenças na representação de dados e como um recurso é acessado.
  - Ocultar arquitetura de máquina
  - Acordo de representação de dados
  - Convenções diferentes em sistemas diferentes

Metas - Transparência

- Localização: esconde a localização física de um recurso.
  - Nomes lógicos. Ex.: URLs
  - http://www.blablabla.com/index.html não deixa claro onde está o servidor ou se foi movido recentemente.
- Migração: recurso pode ser movido para outra localidade sem afetar o modo como é acessado.
  - Ex: URLs: www.abcd.com/index.html - não importa onde estava ou onde está agora, parece igual ao usuário.

Metas - Transparência

- Relocação: recurso pode ser movido para outra localidade enquanto em uso.
  - Ex.: laptops, celulares, migração de máquinas virtuais.
- Replicação: esconde fato de que um recurso é replicado.
  - Réplicas com mesmo nome.
  - Em geral sistema que suporta transparência de replicação deve suportar transparência de localização.
  - Ex.: caches

Metas - Transparência

- Concorrência: esconde que um recurso pode ser compartilhado por diversos usuários competitivos.
  - Compartilhamento pode ser competitivo: acesso a disco, banco de dados, etc.
  - Importante: consistência.
  - Uso de travas de acesso, que dão a cada usuário, um por vez, acesso exclusivo ao recurso desejado.
  - Outro possível mecanismo: transações.

Metas - Transparência

- Falha: esconde falha/recuperação de um recurso.
  - “Você sabe que está usando um sistema distribuído quando uma falha em um computador do qual você nunca ouviu falar impede que você faça qualquer trabalho.” (Leslie Lamport)
  - Transparente à falha: Sistema supera a falha sem percepção do usuário.
  - Mascarar falhas: questão difícil, ou até impossível quando adotadas certas premissas.
    - Problema: diferenciar recursos mortos de recursos lentos.
    - Ex. página web “indisponível”: servidor avariado ou atraso na resposta?

Grau de transparência

- Ocultar todos os aspectos de distribuição pode não ser desejável / possível.
  - Ex.: jornal da manhã por e-mail/celular.
  - Ex. 2: restrição de camada física: limitações do meio físico / processamento em roteadores.

Grau de transparência

- Compromisso: grau de transparência vs. desempenho
  - Tentar mascarar falha transitória pode reduzir a velocidade do sistema como um todo
    - Ex.: repetidas tentativas de acesso antes de desistência.
  - Consistência entre várias réplicas em diferentes continentes: alteração deve propagar-se para todas as cópias antes de novas operações; pode demorar muito e não pode ser ocultado dos usuários.
  - Expor localização
    - com mobilidade, noção de localização pode ser importante.
    - pode ser melhor que ocultar, considerando mobilidade e ubiqüidade.
    - Ex.: impressora no local atual versus impressora em local usual.

Grau de transparência

- Faz sentido tentar alcançar total transparência de distribuição?
- Melhor tornar distribuição explícita
  - Desenvolvedor não consideraria que ela existe.
  - Usuários entenderiam de maneira mais clara o comportamento do sistema distribuído e estariam melhor preparados para lidar com esse comportamento.
- Total transparência nem sempre é vantajoso.
  - Deve ser considerada em conjunto com outras questões como desempenho e facilidade de compreensão.
  - Deve-se levar em conta os propósitos do sistema distribuído.

Metas - Abertura

- Sistema distribuído aberto oferece serviços de acordo com regras padronizadas.
  - Regras descrevem sintaxe e semântica dos serviços.
  - Ex.: mensagens em protocolos de rede.
  - Em SDs: serviços são especificados por meio de interfaces descritas em linguagem de definição de interface (IDL).

Metas - Abertura

- IDLs (interface definition languages):
  - Especifica sintaxe - nomes de funções que estão disponíveis, tipos de parâmetros, valores de retorno, exceções.
  - Semântica: parte difícil. Na prática, informal com língua natural.
  - Interface adequadamente especificada permite que processo arbitrário se comunique com outro processo que fornece aquela interface.
  - Permite construção independente e complementar de sistemas distribuídos.
- Especificações adequadas devem ser completas e neutras
  - Completa: tudo que é necessário para sua implementação foi especificado.
  - Neutra: não prescreve “aparência” da implementação.

Metas - Abertura

- Interoperabilidade: caracteriza até que ponto duas implementações de sistemas ou componentes de fornecedores diferentes devem co-existir e trabalhar em conjunto.
- Portabilidade: caracteriza até que ponto uma aplicação para um SD A pode ser executada, sem modificação, em um SD B que implementa as mesmas interfaces.

Metas - Abertura

- SD aberto – outras características
  - Deve ser de fácil configuração.
  - Deve ser fácil adicionar/substituir componentes sem afetar os que continuam funcionando.
  - Sistema “extensível”.
    - Deve ser fácil adicionar componentes com sistemas operacionais diferentes;
    - Deve ser fácil substituir sistemas de arquivos.
    - Etc.

Abertura – política e mecanismo

- Alcançar flexibilidade: necessário componentes pequenos e de fácil substituição / adaptação.
- Definições não somente para interfaces “externas”, mas também internas.
- Sistemas monolíticos tendem a ser fechados, com separação lógica de componentes, mas que implementam um único programa.
  - Dificulta substituição e adaptação sem afetar todo o sistema.

Abertura – política e mecanismo

- Necessidade de alterar um sistema distribuído: componente que não fornece a política ideal para uma aplicação ou usuário específico.
- Ex:. cache web
  - Usuário pode configurar política de cache (tamanho do cache, freqüência de verificação de consistência).
  - Não consegue tomar decisão baseado em conteúdo do documento, ou qual documento deve ser removido quando cache estiver cheia.
  - Componente com políticas do usuário seria ideal, mas interface com browser é necessária.

Metas - Escalabilidade

- Meta importante devido à popularidade de dispositivos computacionais conectados em rede.
- Escalabilidade em três dimensões:
  - Em tamanho – fácil adicionar usuários e recursos.
  - Em termos geográficos – usuários e recursos podem estar distantes.
  - Em termos administrativos – fácil/possível de gerenciar mesmo abrangendo muitas organizações e recursos.
- Sistema escalável em uma ou mais dimensões frequentemente apresenta perda de desempenho com ampliação.

Metas - Escalabilidade

- Necessidade de ampliar um sistema encontra barreiras de tipos diferentes.
- Escalabilidade em relação ao tamanho:
  - Mais usuários e mais recursos se deparam com o problema de centralização de serviços, dados e algoritmos.
  - Ex.: serviços centralizados implementados em um único servidor em uma máquina específica do sistema distribuído → gargalo
  - Gargalo pode ser de processamento, armazenamento, comunicação.
  - Pode ser inevitável centralizar (ex.: sistemas críticos).

Metas - Escalabilidade

- Serviços centralizados. Ex.: único servidor para todos os usuários.
  - Problema: muitos usuários / aplicações.
  - Pode ser necessário por questões de segurança.
- Dados centralizados.
  - Exs.: um único catálogo de telefones online; DNS centralizado.
  - Armazenamento: factível.
  - Problema: gargalo no acesso – rede e disco.
- Algoritmos centralizados. Ex.: realizar roteamento baseado em informações completas.
  - Problema: colher e transportar toda a informação a cada alteração sobrecarrega a rede.
  - Preferência por algoritmos descentralizados.

Escalabilidade

- Características de algoritmos descentralizados:
  - Nenhuma máquina tem informações completas sobre o estado do sistema.
  - Máquinas tomam decisões baseadas em informações locais.
  - Falha em uma máquina não arruína o algoritmo.
  - Não há suposição implícita que um relógio global existe.

Escalabilidade

- Relógio global:
  - “Precisamente às 12:00:00 todas as máquinas anotarão o tamanho da sua fila de saída.” – precisaria de sincronização exata dos relógios. Quanto maior o sistema, maior a incerteza na sincronização.
- Escalabilidade geográfica
  - Difícil ampliar sistemas de LANs para larga escala distribuída, pois muitas vezes são baseados em comunicação síncrona.
  - Funciona bem quando comunicação é rápida.
  - Comunicação em longa distância: inerentemente não confiável.
  - Localizar um serviço: broadcast em LAN vs. sistema distribuído.

Escalabilidade

- Escalabilidade entre diferentes domínios administrativos
  - Políticas conflitantes de utilização e/ou pagamento.
  - Extensão de um sistema a outros domínios demanda medidas de segurança.
    - Proteger-se contra ataques do novo domínio (ex. acesso somente leitura aos novos usuários)
    - Restringir acesso a componentes e/ou dados críticos.
    - Impor limites de acesso a códigos potencialmente perigosos.
    - Problema: como impor tais limites.

Escalabilidade / técnicas

- Como resolver problemas de escalabilidade?
- Muitas vezes esses problemas aparecem na forma de queda de desempenho com aumento do número de usuários/ componentes.
  - Capacidade limitada de servidores e da rede.

- Três técnicas:
  - Ocultar latências de comunicação (comunicação assíncrona);
  - Distribuição;
  - Replicação.

Escalabilidade / técnicas

- Ocultar latências de comunicação
  - Não aguardar por resposta: comunicação assíncrona.
  - Útil em processamento em lotes e aplicações paralelas nas quais tarefas mais ou menos independentes podem ser escalonadas para execução enquanto outra espera comunicação.

  - Há aplicações onde não é possível fazer comunicação assíncrona
    - ex: aplicações interativas: nada melhor a fazer que esperar a resposta.
    - Solução melhor: reduzir comunicação global.
    - Ex.: verificação de erros de sintaxe em validação de formulário: código no servidor vs. código no cliente. Fig 20.

Escalabilidade / técnicas

- Distribuição
  - Dividir componente e espalhar pelo sistema.
  - Ex.: DNS. Fig 21.; Web.

- Replicação
  - Aumenta disponibilidade
  - Diminui latência
  - Equilibra carga

Escalabilidade / técnicas

- Replicação
  - Ex.: Cache (forma “especial” de replicação);
    - Cache: decisão tomada pelo cliente; replicação pelo proprietário.
    - Cache: sob demanda; replicação: planejada.
    - Problema: consistência.
  - Inconsistência pode ser tolerada dependendo da utilização de um recurso.
    - Cache web: aceitável documento não atualizado por alguns minutos.
    - Bolsas de valores e leilão eletrônico: não aceitável.
    - Forte consistência demanda atualização propagada para todas as outras copias imediatamente → mecanismos de sincronização global.
      - Difícil e ineficiente em sistemas de larga escala.
  - Tolerar inconsistências reduz necessidade de sincronização global, mas isso é dependente de aplicação.

Escalabilidade / técnicas

- Escalabilidade de tamanho em muitos casos é a menos problemática → simples aumento da capacidade de uma máquina resolve a questão, ao menos temporariamente.
- Escalabilidade geográfica mais difícil, depende da natureza.
  - Combinar técnicas de distribuição, replicação e cache com diferentes formas de consistência é suficiente em diversos casos.
- Escalabilidade administrativa é mais difícil porque envolve não só problemas técnicos (políticas de organizações e colaborações humanas).
  - Solução parcial: P2P, onde usuários finais tem o controle administrativo.

Armadilhas

- Suposições falsas comuns feitas por desenvolvedores:
  - Rede é confiável;
  - Rede é segura;
  - Rede é homogênea;
  - A topologia não muda;
  - Latência é zero;
  - Largura de banda é infinita;
  - Custo do transporte é zero;
  - Há um administrador.